

ipd4200mallrmTES-10

**Defense Information Infrastructure (DII)
Common Operating Environment (COE)**

**Application Program Interface Reference Manual (APIRM)
for the
Latitude-Longitude-Time (LLT) Observation
API (MALLT) Segment
of the
Tactical Environmental Support System Next Century
[TESS(NC)]
Meteorology and Oceanography (METOC) Database**

Preliminary Release

Document Version 4.3

9 October 1998

**Prepared for:
Naval Research Laboratory
Marine Meteorology Division
Monterey, CA**

**Prepared by:
Integrated Performance Decisions
Middletown, RI**

PRINTED COPY IS UNCONTROLLED AND MAY BE OBSOLETE

ipd4200malltrmTES-10

Table of Contents

1	SCOPE.....	1
1.1	Identification	1
1.2	System Overview	1
2	REFERENCED DOCUMENTS	5
2.1	Government Documents	5
2.2	Non-Government Documents.....	7
3	MALLT OVERVIEW	9
3.1	Overview of the LLT Database (MDLLT) Segment.....	9
3.2	API Overview	13
3.3	Supersession of TEDS APIs	15
3.4	LLT Observation Data Structures	16
3.4.1	Structures Used for Multiple Observation Types.....	16
3.4.1.1	Generalized Observation Data Structure.....	16
3.4.1.2	Observation Type and Subtype Structure.....	17
3.4.1.3	Station Conditions Structure.....	18
3.4.1.4	Station Information Structure.....	18
3.4.1.5	Observation Reference Structure	18
3.4.1.6	Geographic Point Structure.....	19
3.4.1.7	Observation Time Structure	19
3.4.1.8	Generic ID Structure	19
3.4.1.9	Report Information Structure	19
3.4.1.10	Subtype Information Structure	20
3.4.1.11	Ship Report Structure.....	20
3.4.2	Upper Air Observation Structures.....	20
3.4.2.1	Upper Air Temperature Report Structure	20
3.4.2.2	Upper Air Minimum/Maximum Winds Data Structure.....	20
3.4.2.3	Upper Air Turbulence and Icing Data Structure	21
3.4.2.4	Upper Air Temperature Sounding Structure	21
3.4.2.5	Upper Air Wind Report Structure	21
3.4.2.6	Upper Air Wind Sounding Structure	21
3.4.3	Structures Used for Aircraft Observations.....	22
3.4.3.1	Pilot Report (PIREP) Structure	22
3.4.4	Structures Used for Upper Air Profiles.....	22
3.4.4.1	Upper Air Profile Product Structure	22
3.4.4.2	Upper Air Profile Data Structure	22
3.4.4.3	Convective Conditions Structure	23
3.4.4.4	Evaporation Duct Height Structure.....	23
3.4.4.5	Upper Air Profile Sounding Structure	24
3.4.5	Structures Used for Rocketsonde Observations.....	24
3.4.5.1	Rocketsonde Report Structure	24

3.4.5.2	Rocketsonde Sounding Structure	24
3.4.6	Aerodrome Report Structures	25
3.4.6.1	METAR/SPECI Report Structure	25
3.4.6.2	METAR/SPECI Fields Structure	25
3.4.6.3	Aerodrome Clouds Structure	25
3.4.6.4	Aerodrome Qualifier-Phenomena Structure	26
3.4.6.5	Runway Conditions Structure	26
3.4.6.6	TAF Report Structure	26
3.4.6.7	TAF Fields Structure.....	26
3.4.6.8	TAF Conditions Structure.....	27
3.4.6.9	Aerodrome Forecast Structure	27
3.4.7	Surface Synoptic Observation Structures.....	27
3.4.7.1	Synoptic Observation Report Structure	27
3.4.7.2	Synoptic Summary Structure	27
3.4.7.3	Cloud Data Structure.....	28
3.4.7.4	Ocean Data Structure	28
3.4.8	Ocean Observation Structures.....	29
3.4.8.1	Buoy Report Structure	29
3.4.8.2	Buoy Surface Data Structure.....	29
3.4.8.3	Buoy Sounding Data Structure.....	29
3.4.8.4	Bathythermograph Report Data Structure.....	30
3.4.8.5	Bathythermograph Sounding Level Structure.....	30
3.4.8.6	Ocean Profile Summary Data Structure	30
3.4.8.7	Bottom Loss Data Structure	31
3.4.8.8	ICECAP Data Structure	31
3.4.8.9	Volume Scattering Detail Structure	32
3.4.8.10	Ocean Profile Sounding Structure.....	32
3.4.8.11	Sound Speed-Related Data Structure	32
3.4.9	Query Structures	33
3.4.9.1	LLT Observation Catalog Query Structure	33
3.4.9.2	Geographic Boundaries Structure	33
3.4.10	Catalog Structures	34
3.4.10.1	LLT Observation Catalog Structure.....	34
3.4.10.2	LLT Forecast Catalog Structure	34
3.4.11	Linked List Structure	35
3.4.12	LLT Observation Return Structure	35
4	CONNECT APIs.....	37
4.1	MALLTConnect.....	38
4.2	MALLTDisconnect	39
4.3	MALLTRemoteConnect.....	40
4.4	MALLTRemoteDisconnect	42
4.5	MALLTSetConnection	43

5	RETRIEVAL APIs.....	45
5.1	MALLTCatalog.....	45
5.2	MALLTCatalogObs.....	47
5.3	MALLTGetByID.....	49
5.4	MALLTGetOb	50
5.5	MALLTGetByQuery	51
5.6	MALLTGetObs.....	52
5.7	MALLTGetStationByArea	53
5.8	MALLTGetStationByID	55
5.8.1	OUTPUT PARAMETERS	55
5.9	Observation Types	56
5.10	Observation Subtype	57
6	DATA MANAGEMENT APIs	59
6.1	MALLTDeleteByID	59
6.2	MALLTDeleteOb	60
6.3	MALLTIgest	61
6.4	MALLTIgestOb	63
6.5	MALLTUpdateByID	64
6.6	MALLTUpdateOb	65
6.7	MALLTDeleteByQuery.....	66
7	MALLT UTILITY METHODS AND FUNCTIONS.....	67
7.1	MALLTFreeLL.....	67
7.2	Null Functions	68
7.2.1	Set To NULL Functions.....	68
7.2.2	Functions to Evaluate Whether a Value is NULL	68
8	NOTES	69
8.1	Glossary of Acronyms.....	69

List of Tables

3-1	METOC Database Observation Data Types	9
3-2	LLT Observation Data APIs	14
3-3	MALLT and TEDS API Cross-Reference	15

List of Figures

1-1	TESS(NC) METOC Database Conceptual Organization	3
3-1	MDLLT Observation Classes	12

1 SCOPE

1.1 Identification

This Application Program Interface (API) Reference Manual (APIRM) describes the APIs provided in the Latitude-Longitude-Time (LLT) Observation API (MALLT) segment, Version 4.2.1.0, of the Tactical Environmental Support System Next Century [TESS(NC)] Meteorology and Oceanography (METOC) Database. The MALLT segment provides APIs for the storage, retrieval, and manipulation of point meteorological and oceanographic observations. These are observations that are taken at a point in space and time, and are therefore characterized by latitude, longitude, and time. This software is designed to run under the Defense Information Infrastructure (DII) Common Operating Environment (COE), release 3.1, on a Hewlett-Packard computer running HP-UX 10.20 or a personal computer running the Microsoft Windows NT 4.0 operating system with Service Pack 3.

1.2 System Overview

The APIs described in this document form a portion of the METOC Database component of the TESS(NC) Program (Navy Integrated Tactical Environmental Subsystem (NITES) Version I). On 29 October 1996, the Oceanographer of the Navy issued a TESS Program Policy statement in letter 3140 Serial 961/6U570953, modifying the Program by calling for five seamless software versions that are DII COE compliant, preferably to level 5.

The five versions are:

- NITES Version I The local data fusion center and principal METOC analysis and forecast system (TESS(NC))
- NITES Version II The subsystem on the Joint Maritime Command Information System (JMCIS) or Global Command and Control System (GCCS) (NITES/Joint METOC Segment (JMS))
- NITES Version III The unclassified aviation forecast, briefing and display subsystem tailored to Naval METOC shore activities (currently satisfied by the Meteorological Integrated Data Display System (MIDDS))
- NITES Version IV The Portable subsystem composed of independent PCs/workstations and modules for forecaster, satellite, communications, and Integrated Command, Control, Communications, Computer, and Intelligence Surveillance Reconnaissance (IC4ISR) functions (currently the Interim Mobile Oceanographic Support System (IMOSS))
- NITES Version V Foreign Military Sales (currently satisfied by the Allied Environmental Support System (AESSION))

NITES I acquires and assimilates various METOC data for use by US Navy and Marine Corps weather forecasters and tactical planners. NITES I provides these users with METOC data, products, and applications necessary to support the warfighter in tactical operations and decision making. NITES I provides METOC data and products to NITES I and II applications, as well as non-TESS(NC) systems requiring METOC data, in a heterogeneous, networked computing environment.

The TESS(NC) Concept of Operations and system architecture require that the METOC Database be distributed both in terms of application access to METOC data and products and in terms of physical location of the data repositories. The organizational structure of the database is influenced by these requirements, and the components of this distributed database are described below.

In accordance with DII COE database concepts, the METOC Database is composed of six DII COE-compliant *shared database* segments. Associated with each shared database segment is an API segment. The segments are arranged by data type as follows:

<u>Data Type</u>	<u>Data Segment</u>	<u>API Segment</u>
Grid Fields	MDGRID	MAGRID
LLT Observations	MDLLT	MALLT
Textual Observations and Bulletins	MDTXT	MATXT
Remotely Sensed Data	MDREM	MAREM
Imagery	MDIMG	MAIMG
Climatology Data	MDCLIM	MACLIM

A typical client-server installation is depicted in Figure 1-1 on the next page. This shows the shared database segments residing on a DII COE SHADE database server, with a NITES I or II client machine hosting the API segments. Communication between API segments and shared database segments is accomplished over the network using ANSI-standard Structured Query Language (SQL).

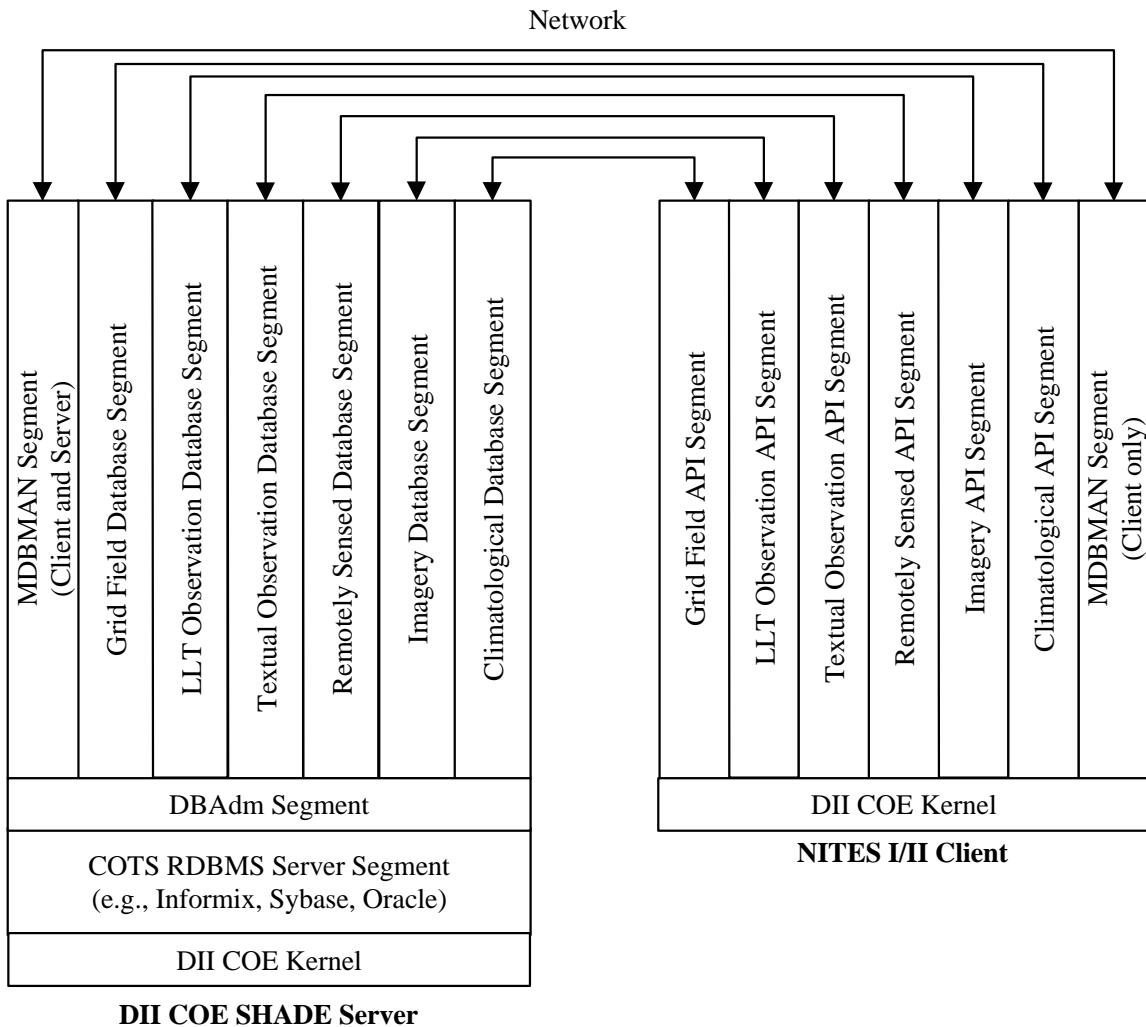


Figure 1-1. TESS(NC) METOC Database Conceptual Organization

The APIs in the MALLT segment deal with point observations. These include surface weather observations (hourlies, specials, synoptic observations, METAR reports, Terminal Aerodrome Forecasts (TAFs), etc.), upper air observations (e.g., radiosonde reports, aircraft observations), and ocean soundings (bathythermograph, sound velocity profiles, etc.). For upper air and ocean soundings, the database may also store data derived from the original soundings in the form of upper air profiles and ocean profiles.

(This page intentionally left blank.)

2 REFERENCED DOCUMENTS

2.1 Government Documents

STANDARDS

MIL-STD-498 *Software Development and Documentation*
5 December 1994

SPECIFICATIONS

DII COE I&RTS July 1997	<i>Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification, Preliminary, Version 3.0</i>
Unnumbered 5 December 1997	<i>Performance Specification (PS) for the Tactical Environmental Support System/Next Century TESS(3)/NC (AN/UMK-3)</i>
Unnumbered 30 September 1997	<i>Software Requirements Specification for the Tactical Environmental Support System/Next Century [TESS(3)/NC] Meteorological and Oceanographic (METOC) Database, Space and Naval Warfare Systems Command, Environmental Systems Program Office (SPAWAR PMW-185), Washington, DC</i>

OTHER DOCUMENTS

Unnumbered 30 September 1997	<i>Database Design Description for the Tactical Environmental Support System/Next Century [TESS(3)/NC] Meteorological and Oceanographic (METOC) Database, Space and Naval Warfare Systems Command, Environmental Systems Program Office (SPAWAR PMW-185), Washington, DC</i>
DII.COE.DocReqs-5 29 April 1997	<i>Defense Information Infrastructure (DII) Common Operating Environment (COE) Developer Documentation Requirements, Version 1.0</i>

Office of the Assistant Secretary of Defense (Command, Control, Communications, and Intelligence), Washington, DC

DoD 8320.1-M-1
19 November 1996

Data Standardization Procedures (Draft)

Commander, Naval Meteorology and Oceanography Command

COMNAVMETOCOMINST 3141.2 *Surface METAR Observations Users Manual*

COMNAVMETOCOMINST 3144.1D *Ship Weather Observations Manual*

Department of the Air Force, Headquarters Air Weather Service, Scott AFB, ILL

AWSR 105-2
24 August 1990

Weather Communications Policies and Procedures

Naval Research Laboratory, Marine Meteorology Division, Monterey, CA

ipd4200mallpmTES-10 *Programming Manual (PM) for the Latitude-Longitude-Time (LLT) Observation API (MALLT) Segment of the Tactical Environmental Support System Next Century [TESS(NC)] Meteorology and Oceanography (METOC) Database*
9 October 1998

ipd4200malltipTES-10 *Installation Procedures (IP) for the Latitude-Longitude-Time (LLT) Observation API (MALLT) Segment of the Tactical Environmental Support System Next Century [TESS(NC)] Meteorology and Oceanography (METOC) Database*
9 October 1998

Space and Naval Warfare Systems Command, Environmental Systems Program Office (SPAWAR PMW-185), Washington, DC

Unnumbered
20 June 1997

*Tactical Environmental Data System (TEDS) Release 3.5
Observation/Profile Data Applications Program Interface (API) User's Guide*

Office of the Federal Coordinator for Meteorological Services and Supporting Research,
Washington, DC

FMH-1 December 1995	<i>Federal Meteorological Handbook No. 1: Surface Weather Observations and Reports</i>
FMH-2 December 1988	<i>Federal Meteorological Handbook No. 2: Surface Synoptic Codes</i>
FMH-10 December 1988	<i>Federal Meteorological Handbook No. 10: Meteorological Rocket Observations</i>
FMH-11 June 1991	<i>Federal Meteorological Handbook No. 11: Doppler Radar Meteorological Observations</i>

2.2 Non-Government Documents

World Meteorological Organization, Geneva, Switzerland

WMO-306 1995	<i>Manual On Codes</i>
WMO-386 1992	<i>Manual on the Global Telecommunications System</i>

(This page intentionally left blank.)

3 MALLT OVERVIEW

3.1 Overview of the LLT Observation Database (MDLLT) Segment

The LLT Observation database was designed using an object/relational hybrid method. The observation data are modeled as classes derived from a common root class. However, the methods are not associated with each class; they are associated with the LLT Observation database in general. The purpose of the LLT Observation APIs is to manage all aspects of decoded meteorological observations from various sources (AWN, GTS, SMOOS) and formats (BUFR, WMO Textual Formats). Table 3-1 describes the decoded messages that MDLLT is designed to handle.

Table 3-1. METOC Database Observation Data Types

Type	Description	WMO Alphanumeric Code	WMO BUFR Sequence
Fixed Surface Station Synoptic Reports	Synoptic report from surface stations reported at regular intervals.	FM-12	3-07-001 3-07-002 3-07-003 3-07-004 3-07-005 3-07-006 3-07-007 3-07-008 3-07-009
Ship Synoptic Reports	Synoptic report from ships reported at regular intervals.	FM-13	3-08-004
Mobile Surface Station Synoptic Reports	Synoptic reports from mobile land stations reported at regular intervals.	FM-14	No equivalent
METAR/SPECI	Aviation Routine Weather Reports and Aviation Selected Special Weather Report.	FM-15, FM-16	3-07-011
Fixed Buoy Report	Reports from fixed ocean buoys.	FM-18	3-08-001 3-08-002
Drifting Buoy Report	Reports from drifting ocean buoys.	FM-18	3-08-003

Table 3-1. METOC Database Observation Data Types

Type	Description	WMO Alphanumeric Code	WMO BUFR Sequence
Upper Air Winds Reported at pressure levels from fixed land station	Upper air report from a surface station that reports only wind information at standard and significant isobaric levels that was taken from a fixed land station.	FM-32	3-09-003 3-09-004
Upper Air Winds Reported at pressure levels from sea station	Upper air report from a surface station that reports only wind information at standard and significant isobaric levels that was taken from a sea station.	FM-33	3-09-012
Upper Air Winds Reported at pressure levels from a mobile land station	Upper air report from a surface station that reports only wind information at standard and significant isobaric levels that was taken from a mobile land station.	FM-34	3-09-016 3-09-019
Upper Air Temperature Report at pressure levels from a fixed land station	Upper air report from a surface fixed land station that reports wind and temperature information at standard and significant isobaric levels.	FM-35	3 09 005 3 09 006 3 09 007 3 09 008
Upper Air Temperature Report at pressure levels from a ship station	Upper air report from a ship station that reports wind and temperature information at standard and significant isobaric levels.	FM-36	3 09 013 3 09 014 3 09 017 3 09 018
Upper Air Temperature Report at pressure levels from a dropsonde	Upper air report from a sonde dropped from a balloon or aircraft station that reports wind and temperature information at standard and significant isobaric levels.	FM-37	No equivalent
Upper Air Temperature Report at pressure levels from a fixed land station	Upper air report from a surface fixed land station that reports wind and temperature information at standard and significant isobaric levels.	FM-38	No equivalent

Table 3-1. METOC Database Observation Data Types

Type	Description	WMO Alphanumeric Code	WMO BUFR Sequence
Upper Air Winds Reported at heights from a surface station	Upper air reports from a surface station that reports only wind information at geopotential heights.	FM-39	3-09-001 3-09-002 3-09-003 3-09-004
Upper Air Winds Reported at heights from a ship	Upper air reports from a ship station that reports only wind information at geopotential heights.	FM-40	3-09-011
Upper Air Profile	Stores an upper air report plus derived data (refractivity profiles, stability indices, etc.)	N/A	N/A
Reports from Aircraft	Reports from aircraft.	FM-41, FM-42 ICAO Aireps	03-11-001
Aerodrome Forecast	Reports and forecast from airfields.	FM-51	
Bathy Report	Report of a bathythermal observation.	FM-63	3-15-001
TESAC	Temperature, Salinity, and current reports from a sea station.	FM-64	3-15-002

At the root class minimal information is available within each class:

- Time
- Latitude
- Longitude
- Observation Type
- Observation Subtype
- Originator
- Classification
- Source Communication Channel
- Air Temperature
- Wind Speed
- Wind Direction
- Pressure.

Leaf node classes model decoded messages and contain all data that were received from the decoded message to allow “re-assembly” of messages. Figure 3-1 illustrates the class hierarchy for LLT observation data.

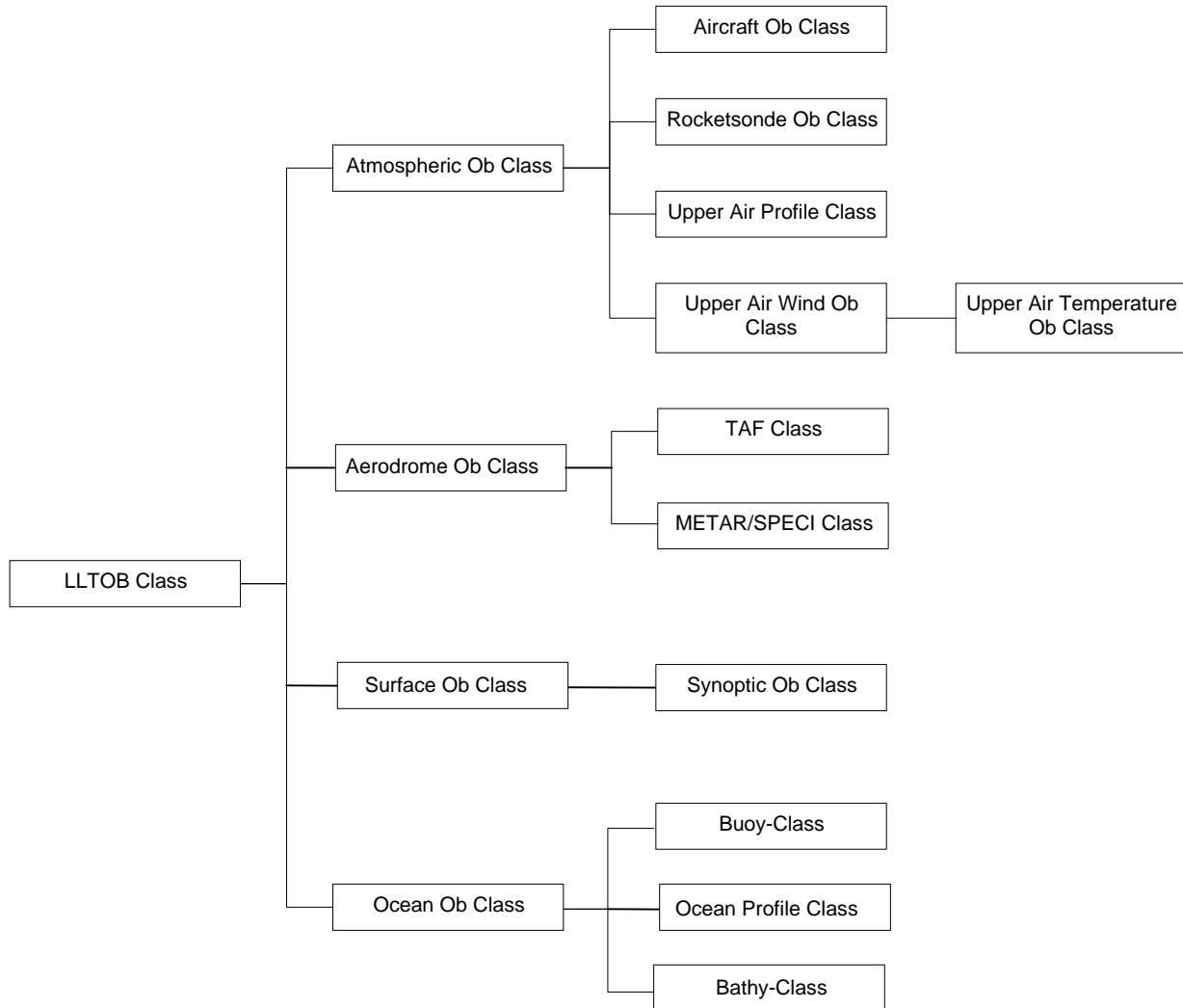


Figure 3-1. MDLLT Observation Classes

3.2 API Overview

The methods are associated with the database in general and not each class. The purpose of this is to use the relational functionality provided by the Relational Database Management System (RDBMS) to implement the data model. There are eight methods (some of whose names have been changed since the preliminary (developer) release). In the list that follows, the new routine name is given first, then the old routine name. Applications using the developer's release routine names should be upgraded to the new routine names, as the old names will be phased out in the next release. For this release, both routine names will continue to function. In addition, two new APIs have been added to perform transformations between station IDs and geographic area by querying the appropriate table in the MDLLT database.

1. **MALLTIgest, MALLTIgestOb:** Adds an object to the database.
2. **MALLTDeleteByID, MALLTDeleteOb:** Deletes objects from the database based on a given criteria.
3. **MALLTCatalog, MALLTCatalogObs:** Provide summary information about observations in the database.
4. **MALLTGetByID, MALLTGet:** Retrieve a single object returned as a result of a catalog.
5. **MALLTGetByQuery, MALLTGets:** Retrieve multiple objects.
6. **MALLTUpdateByID, MALLTUpdateOb:** Updates limited attributes within an object.
7. **MALLTDeleteByQuery:** Allows purging from the database of observations matching query criteria.
8. **MALLTFreeLL:** Used to free a linked list returned by a query operation.
9. **MALLTGetStationByArea:** Receives a geographic area, queries the database for stations residing there, then returns a list of the stations found.
10. **MALLTGetStationByID:** Receives a station ID and returns detailed station information.

Each of the methods, with the exception of the Ingest methods, allows the input of selection criteria. The criteria are used to construct SQL statements to retrieve or otherwise manipulate objects that match the criteria. The Ingest methods do not provide any criteria because they are simply writing data to the database.

Five other methods are also provided to manage database connections:

1. **MALLTConnect:** Connect to the default LLT database.
2. **MALLTRemoteConnect:** Connect to the specified LLT database.

3. **MALLTDisconnect:** Disconnect from the default LLT database.
4. **MALLTRemoteDisconnect:** Disconnect from the specified LLT database.
5. **MALLTSetConnection:** Set the current connection to the specified LLT database.

Table 3-2 provides a list of all APIs for LLT observation data.

Table 3-2. LLT Observation Data APIs

API Name	Functional Description
MALLTCatalog, MALLTCatalogObs	Catalogs observations based on a selection criteria.
MALLTConnect	Connects to the default LLT Observation Database.
MALLTDeleteByID, MALLTDeleteOb	Deletes an observation from the database.
MALLTDisconnect	Disconnects from the LLT Observation Database.
MALLTGetByID, MALLTGetOb	Gets a single observation based on information returned from a catalog query.
MALLTGetByQuery, MALLTGetObs	Retrieves groups of observations based on selection criteria.
MALLTGetStationByArea, MALLTGetStationByID	Gets station IDs and geographic area by querying the MDLLT database.
MALLTIgest, MALLTIgestOb	Adds an observation to the database.
MALLTFreeLL	Frees a linked list returned by a query operation.
MALLTDeleteByQuery	Purges observations from the database based on selection criteria.
MALLTRemoteConnect	Connects to the specified LLT Observation Database.
MALLTRemoteDisconnect	Disconnects to the specified LLT Observation Database.
MALLTSetConnection	Sets the active connection to the specified LLT Observation Database.
MALLTUpdateByID, MALLTUpdateOb	Updates an observation based on given criteria.

3.3 Supersession of TEDS APIs

The MALLT APIs will, when implemented, replace the older TEDS APIs, documented in the *TEDS Release 3.5 Observation/Profile API User's Guide* referenced in Section 2 of this document. Applications using the TEDS APIs will need to be rewritten to use the equivalent MALLT APIs. Table 3-3 provides a cross-reference of the MALLT and TEDS APIs for the convenience of developers.

The following important considerations should be noted:

- In some cases, a single MALLT API replaces several TEDS APIs.
- Some of the APIs in the *TEDS Release 3.5 Observation/Profile API User's Guide* have been placed in other segments of the TESS(NC) METOC Database. The APIs for textual observations and bulletins are in the MATXT segment, and those for vertical soundings and Special Sensor Microwave/Imager will be in the MAREM segment.
- Because the data structures differ between the two implementations, it is imperative that care be taken to ensure correct conversion.

Table 3-3. MALLT and TEDS API Cross-Reference

TEDS API(s)	MALLT API
ted_AddAC, ted_AddBT, ted_AddOceanProfile, ted_AddSN, ted_AddUA, ted_AddUP	MALLTIgest
ted_Add2ACDS, ted_Add2BTDS, ted_Add2OPDS, ted_Add2SNDS, ted_Add2UADS, ted_Add2UPDS	MALLTIgest
ted_GetACCatalog, ted_GetBTCatalog, ted_GetOceanProfileCatalog, ted_GetSNCatalog, ted_GetUACatalog, ted_GetUPCatalog	MALLTCatalog
ted_start	MALLTConnect
ted_stop	MALLTDisconnect
ted_GetAC, ted_GetBT, ted_GetOceanProfile, ted_GetSN, ted_GetUA, ted_GetUP	MALLTGetByID
ted_GetACs, ted_GetBTs, ted_GetOceanProfiles, ted_GetSNs, ted_GetUAs, ted_GetUPs	MALLTGetByQuery
ted_DeleteAC, ted_DeleteBT, ted_DeleteOceanProfile, ted_DeleteSN, ted_DeleteUA, ted_DeleteUP	MALLTDeleteByID
ted_UpdateAC, ted_UpdateBT, ted_UpdateOceanProfile, ted_UpdateSN, ted_UpdateUA, ted_UpdateUP	MALLTUpdateByID

3.4 LLT Observation Data Structures

The structures described in the following subsections are used by one or more of the LLT Observation APIs. They are provided here for reference.

3.4.1 Structures Used for Multiple Observation Types

3.4.1.1 Generalized Observation Data Structure

This structure, defined in the file *MALLT.h*, holds data from a single observation of any supported type. Substructures are defined in the sections that follow.

```
typedef struct tagObservation {
    MALLTOBTYPES          obType;      /* Type and subtype of ob */
    MALLTGENERALCONDITIONS gc;         /* Conditions at reporting station */
                                         /* Also used for Aircraft reports */
                                         /* Not used by METAR/SPECI or TAF */
    MALLTOBREFERENCE       obRef;       /* Internal database reference info */
    MALLTGEOPPOINT         geoLoc;     /* Geographic location */
                                         /* Not used by METAR/SPECI, TAF,
                                         /* or fixed land station reports */
                                         /* on ingest; they are set when
                                         /* queried
    MALLTMETOCTIME        baseTime;    /* Time of report */
    MALLTGENERICID         reporterID; /* ICAO call sign, Ship ID, or
                                         /* WMO ID
    MALLTREPORTINFO        reportInfo; /* QC value, method of receipt,
                                         /* data category
    MALLTSUBTYPEINFO       sti;        /* Information received based on
                                         /* observation subtype
                                         /* Data specific to a report type
    union {
        MALLTPIREP          pirep;      /* Pilot report data
        MALLTMETARREPORT     metar;      /* METAR/SPECI report data
        MALLTTAFREPORT        TAF;        /* TAF report data
        MALLTBUOYREPORT      buoy;      /* Buoy observation data
        MALLTBATHYREPORT     bathy;      /* Bathythermograph data
        MALLTSYNOPTICREPORT  synoptic;   /* Synoptic report data
        MALLTUATEMPREPORT    uaTemp;    /* Upper air temps
        MALLTUAWINDREPORT    uaWind;    /* Upper air winds
        MALLTUAPPRODUCT      uaPPProd;  /* Upper air profile data
        MALLTROCKETSONDEREPORT rocket;   /* Rocketsonde report data
        MALLTOCEANPROFILE     oceanProfile; /* Ocean profile data
    } ob;
} MALLTOBSERVATION, *PMALLTOBSERVATION;
```

3.4.1.2 Observation Type and Subtype Structure

This structure contains the observation type and subtype codes.

```
typedef struct MALLTObTypes {
    short sObType,                      /* Observation type           */
          sObSubType;                   /* Observation subtype        */
} MALLTOBTYPES, *PMALLTOBTYPES;
```

The observation types and subtypes are defined in the file *malltypes.h* as follows (report forms from WMO-306 are cited where they apply, but it should be noted that the same data may be received in other formats):

```
/* Ob Data Types */
#define MALLT_UPPERAIR_WIND      1  /* Upper wind report (WMO FM-32/33/34)   */
#define MALLT_UPPERAIR_TEMP      2  /* Upper temp report (WMO FM-35 thru 39) */
#define MALLT_ROCKET_OB          3  /* Rocketsonde rpt (WMO FM-39/40)         */
#define MALLT_SYNOPTIC_OB        4  /* Synoptic report (WMO FM-12/13/14)       */
#define MALLT_OCEAN_OB           5  /* Ocean ob (WMO FM-18/63/64)             */
#define MALLT_AIRCRAFT_OB        6  /* Aircraft report (WMO FM-41/42, PIREP)  */
#define MALLT_AERODROME_OB       7  /* TAF (FM-51, METAR/SPECI (FM-15/16)     */

/* Ocean Ob Data Subtypes */
#define MALLT_BATHY_REPORT       1  /* Reports from bathythermograph,           */
                                /* sound velocimeter, and similar depth   */
                                /* profiling instruments                  */
#define MALLT_BUOY_REPORT        2  /* Buoy reports                           */
#define MALLT_OCEAN_PROFILE      14 /* Ocean profiles (may contain computed  */
                                /* data in addition to the data          */
                                /* initially reported                   */

/* AERODROME Ob Data Sub Types */
#define MALLT_METAR_REPORT       3  /* Report in WMO FM-15 METAR format      */
#define MALLT_SPECI_REPORT        4  /* Report in WMO FM-16 SPECI format      */
#define MALLT_TAF_REPORT          5  /* Report in WMO FM-51 TAF format        */

/* Upper Air(TEMP and WIND), Rocket and Synoptic Ob Data Sub Types */
#define MALLT_SHIP                6  /* Reports from ships (FM-13/33/36/40)   */
#define MALLT_FIXED_LAND          7  /* Reports from fixed land stations      */
                                /* (FM 12/32/35/39)                     */
#define MALLT_MOBILE_LAND         8  /* Reports from mobile land stations     */
                                /* (FM-14/34/38)                       */

/* Upper Air Temp subtypes*/
#define MALLT_UAT_DROP            9  /* Dropsonde report (FM-37)              */
#define MALLT_UAT_PROFILE         10 /* Upper air temperature profile (may   */
                                /* contain calculated data in addition */
                                /* to the data initially reported)     */

/* Aircraft OB subtypes*/
#define MALLT_AIRREP              12 /* AIREP report                          */
#define MALLT_PIREP                13 /* PIREP report                          */
```

Four new station information macros have been included in the *malltmacros.h* header file. They are used to identify the type of station information desired and are declared as follows:

```
#define MALLT_WMOSTATION      1      /* Get information about WMO stations      */
#define MALLT_ICAOSTATION       2      /* Get information about ICAO stations      */
#define MALLT_BUOY               4      /* Get information about Buoys by ID        */
#define MALLT_ALL                7      /* Get information about all stations       */
```

3.4.1.3 Station Conditions Structure

This structure contains information about the conditions at the reporting station.

```
typedef struct MALLTStationConditions {
    float   rsAirTemperature;      /* Air temperature (degrees C or F)          */
    float   rsWindSpeed;          /* Wind speed (meters/second or knots)       */
    short   sWindDirection;       /* Direction from which wind is blowing      */
                                         /* (degrees clockwise from true north)        */
    float   rsPressure;          /* Air pressure (hectopascals or inches       */
                                         /* of mercury)                            */
    float   rsAltitude;          /* Station elevation (or altitude for       */
                                         /* aircraft reports)                      */
} MALLTGENERALCONDITIONS, *PMALLTGENERALCONDITIONS;
```

3.4.1.4 Station Information Structure

The **MALLTStationInfo** structure (typedef **MALLTSTATIONINFO**) is defined in the *malltcommon.h* header file.

```
typedef struct tagMALLTStationInfo
{
    int      nWMOID;           /* WMO station identifier                  */
    char     szICAOloc[8];     /* ICAO callsign                         */
    char     szPlaceName[64];   /* Common name of station location        */
    char     szState[4];       /* State where station resides            */
    char     szCountry[32];    /* Country where station resides          */
    char     szWMORegion[4];   /* WMO station region                    */
    float   rsStationLat;     /* Latitude of station                   */
    float   rsStationLon;     /* Longitude of station                  */
    float   rsUALat;          /* Upper Air latitude (sonde location)   */
    float   rsUALon;          /* Upper Air longitude (sonde location)  */
    float   rsStnElev;        /* Elevation of station                  */
    float   rsUAElev;         /* Elevation of Upper Air (sonde)        */
    char     szRBSN[4];        /* Regional Basic Synoptic Network no.   */
    int      nStationType;    /* Type of station, WMO, or ICAO          */
} MALLTSTATIONINFO, *PMALLTSTATIONINFO;
```

3.4.1.5 Observation Reference Structure

This structure contains internal database reference information about an observation.

```
typedef struct MALLTObRef {
    int      nObjectID;        /* Unique object identifier                 */
    char     szTableName[19];   /* Name of database table where stored   */
} MALLTOBREFERENCE, *PMALLTOBREFERENCE;
```

3.4.1.6 Geographic Point Structure

This structure contains information about the geographic location at which an observation was collected.

```
typedef struct MALLTGeoPoint {
    float    rsLatitude;           /* Latitude of observation point (positive      */
                                /* north, negative south) (-90.0 to 90.0)   */
    float    rsLongitude;          /* Longitude of observation (positive        */
                                /* east, negative west) (-180.0 to 180.0)   */
} MALLTGEOPPOINT, *PMALLTGEOPPOINT;
```

3.4.1.7 Observation Time Structure

This structure contains observation time data.

```
typedef struct MALLTMetocTime {
    char     szTime[32];          /* String representation of the time in a      */
                                /* format dependent on the ob type          */
    long     lTime;               /* Epoch time (time in seconds since 0000Z   */
                                /* 1 January 1970)                         */
} MALLTMETOCTIME, *PMALLTMETOCTIME;
```

3.4.1.8 Generic ID Structure

This structure contains identification information common to all observation types.

```
typedef struct MALLTGenericID
{
    char     szSiteName[8];       /* ICAO call sign or ship international call */
                                /* sign of reporting station                  */
    int      nSiteID;            /* WMO block station number or buoy platform */
                                /* ID of reporting station                   */
} MALLTGENERICID, *PMALLTGENERICID;
```

3.4.1.9 Report Information Structure

This structure holds data about a report, including the method of receipt, quality control value, and flags.

```
typedef struct MALLTObInfo{
    int      nQualityFlag;        /* Quality control indicator -- 0 if no        */
                                /* quality control checks were done          */
    int      nDataCatagory;       /* Code designating state of data relative to */
                                /* its original form: 0 = base, 1 = edited,   */
                                /* 2 = derived                               */
    char    szReceiptMethod[32];  /* Receipt method (e.g. COMEDS)                */
}MALLTREPORTINFO, *PMALLTREPORTINFO;
```

3.4.1.10 Subtype Information Structure

This structure holds report information peculiar to a particular type of report.

```
typedef union {
    char *szInstrumentation;           /** Used with fixed sea stations      */
    MALLTSHIPREPORT ship;             /** Used with Ships                  */
} MALLTSUBTYPEINFO, *PMALLTSUBTYPEINFO;
```

3.4.1.11 Ship Report Structure

This structure contains information peculiar to ship reports.

```
typedef struct MALLTShipReport {
    int      rsShipSpeed;            /* Ship's speed in knots          */
    int      rsShipDirection;        /* Ship's heading in degrees clockwise from
                                    /* true north                      */
} MALLTSHIPREPORT, *PMALLTSHIPREPORT;
```

3.4.2 Upper Air Observation Structures

3.4.2.1 Upper Air Temperature Report Structure

This structure contains data from an upper air temperature, humidity, and wind report.

```
typedef struct MALLTUATempReport {
    MALLTUAMMWINDS      ummw;       /* Minimum/Maximum winds data structure */
    MALLTUATTI         uti;        /* Turbulence and icing data structure */
    MALLTCLOUDDATA     clouds;     /* Cloud data structure */
    int                nNumLevels;  /* Number of levels reported */
    MALLTUATEMPSOUNDING *pUATS;    /* Pointer to structures containing data
                                    /* for each reported level           */
} MALLTUATEMPREPORT, *PMALLTUATEMPREPORT;
```

3.4.2.2 Upper Air Minimum/Maximum Winds Data Structure

This structure contains data concerning the minimum and maximum winds in a sounding.

```
typedef struct MALLTUAMMaxWinds {
    float   rsWind1KMAboveMax;      /* Vector difference between maximum
                                    /* wind and wind blowing 1 km above the
                                    /* level of maximum wind           */
    float   rsWind1KMBelowMax;      /* Vector difference between maximum
                                    /* wind and wind blowing 1 km below the
                                    /* level of maximum wind           */
    short   sIRCorrection;         /* Identifier for the IR correction used */
    short   sRadioSond;            /* Identifier for the radiosonde type */
    short   sTrackingSystem;       /* Identifier for the radiosonde
                                    /* tracking system used           */
} MALLTUAMMWINDS, *PMALLTUAMMWINDS;
```

3.4.2.3 Upper Air Turbulence and Icing Data Structure

This structure contains data concerning turbulence and icing at levels above the surface.

```
typedef struct MALLTUATTurbulenceIce {
    MALLTMETOCTIME launchTime;          /* Launch time of sensor */
    float           rsSst;              /* Sea surface temperature */
    char            szHeightLowestTurb; /* Lowest level at which
                                         /* turbulence reported */
    short           sHeightLowestIce;   /* Lowest level at which icing
                                         /* reported */
} MALLTUATTI, *PMALLTUATTI;
```

3.4.2.4 Upper Air Temperature Sounding Structure

This structure holds data from a single level of an upper air temperature, humidity, and wind sounding.

```
typedef struct MALLTUATempSounding {
    float   rsHeight;                /* Height of level in meters, hectopascals,
                                         /* or geopotential meters (dependent on
                                         /* szHeightType) */
    float   rsPressure;              /* Air pressure at level in hectopascals */
    float   rsWindSpeed;             /* Wind speed at level in m/s or knots */
    float   rsAirTemperature;        /* Air temperature at level in degrees C */
    float   rsMoisture;              /* Moisture value at level (dependent on
                                         /* szMoistureType) */
    float   rsWindDirection;         /* Wind direction at level (degrees clock-
                                         /* wise from true north) */
    short   sHeightType;             /* Indicates units of rsHeight. 0 indicates
                                         /* meters, 1 hectopascals, 2 geopotential
                                         /* meters */
    char    szMoistureType;          /* Indicates how moisture is reported. 0
                                         /* indicates that rsMoisture reports
                                         /* relative humidity, 1 indicates dew point */
} MALLTUATEMPSOUNDING, *PMALLTUATEMPSOUNDING;
```

3.4.2.5 Upper Air Wind Report Structure

This structure holds data from an upper air wind report.

```
typedef struct MALLTUAWReport {
    MALLTUAMMWINDS      ummw;        /* Min-max winds structure (Sec. 3.4.11) */
    int                 nNumLevels;  /* Number of levels reported */
    MALLTUAWINDSOUNDING *pUAWS;     /* Pointer to UA Wind Sounding
                                         /* structures containing data for levels */
} MALLTUAWINDREPORT, *PMALLTUAWINDREPORT;
```

3.4.2.6 Upper Air Wind Sounding Structure

This structure contains upper air wind data for a single sounding level.

```
typedef struct MALLTUAWindSounding {
    float   rsHeight;                /* Height of level in meters, hectopascals,
                                         /* or geopotential meters (dependent on
                                         /* szHeightType) */
    float   rsPressure;              /* Pressure at level in hectopascals */
    float   rsWindSpeed;             /* Wind speed at level */
    float   rsWindDirection;         /* Wind direction at level (degrees clock-
                                         /* wise from true north) */
} MALLTUAWINDSOUNDING, *PMALLTUAWINDSOUNDING;
```

```
    /* wise from true north) */  
short sHeightType; /* Indicates units of rsHeight. 0 indicates */  
/* meters, 1 hectopascals, 2 geopotential */  
/* meters */  
} MALLTUAWINDSOUNDING, *PMALLTUAWINDSOUNDING;
```

3.4.3 Structures Used for Aircraft Observations

3.4.3.1 Pilot Report (PIREP) Structure

The PIREP structure contains meteorological and operational data reported by an aircraft in flight.

```
typedef struct MALLTPiRep {  
    char szRemarks[32]; /* Remarks */  
    float rsMoisture; /* Dew point temperature or relative humidity, */  
    /* as indicated by szMoistureType */  
    char szPhaseOfFlight[4]; /* Phase of flight (LVR=routine observation in */  
    /* level flight, LVW=highest wind encountered */  
    /* in level flight, ASC=observation during */  
    /* ascent, DES=observation during descent */  
    char szAirframeIcing[4]; /* Airframe icing code */  
    char szMoistureType[2]; /* 0 if szMoisture is relative humidity, 1 if */  
    /* szMoisture is dew point temperature */  
    char szNavSystem[2]; /* Code for type of navigation system used */  
    short sTurbulence; /* Code for turbulence type reported */  
    short sTurbulenceTop; /* Highest altitude of turbulence reported */  
} MALLTPIREP, *PMALLTPIREP;
```

3.4.4 Structures Used for Upper Air Profiles

3.4.4.1 Upper Air Profile Product Structure

This structure stores an upper air sounding and information derived from that sounding. This allows applications to calculate data based on the profile once and then store them in the database for future use.

```
typedef struct MALLTUAPPProduct{  
    MALLTUAPROFILE uap; /* Upper Air Profile data structure */  
    MALLTCNVCOND convCloud; /* Convective cloud data structure */  
    int nNumProfiles; /* Number of profiles stored */  
    MALLTEVAPORATIONHT evapHT; /* Evaporative duct height */  
    MALLTUAPROFILESOUNDING *pUAPS; /* Pointer to upper air profile */  
    /* sounding structures for all */  
    /* levels */  
} MALLTUAPPDIRECT, *PMALLTUAPPDIRECT;
```

3.4.4.2 Upper Air Profile Data Structure

This structure holds information calculated from the upper air observation data. The data contained here are not level specific (i.e., there is no value for each level in the profile).

```
typedef struct MALLTUAProfile {  
    int nHeightContrailBPRB; /* Height of the bottom of the layer in */  
    /* which contrail formation is probable */  
    int nHeightContrailTPRB; /* Height of the top of the layer in which */
```

```

int      nHeightContrailBPO;          /* contrail formation is probable */ 
int      nHeightContrailTPO;          /* Height of the bottom of the layer in */
                                    /* which contrail formation is possible */ 
int      nEquilLevel;                /* Height of the top of the layer in which */
                                    /* contrail formation is possible */ 
int      nEquilTemp;                 /* Height at which the temperature of a */
                                    /* buoyantly rising parcel again becomes */
                                    /* equal to the temperature of the */
                                    /* environment (equilibrium level) */ 
int      nLiftedIndex;               /* Temperature of a buoyantly rising parcel */
                                    /* as it becomes equal to the temperature */
                                    /* of the environment */ 
float   rsPrecipWatCon;             /* Precipitable water content */ 
int      nKIndex;                   /* Lifted Index */ 
int      nShowwalterIndex;           /* K Index */ 
int      nShowwalterIndex;           /* Showalter Index */ 
int      nSweatIndex;                /* Severe Weather Threat (SWEAT) Index */ 
int      nVertIndex;                /* Vertical Totals Index */ 
int      nCrossIndex;                /* Cross Totals Index */ 
int      nTotalIndex;                /* Total Totals Index */ 
int      nWetBulbZHeight;            /* Height of wet bulb temp measurement */ 
int      nMaxConWindGustDir;         /* Direction of maximum wind gust */ 
float   rsMaxConWndGustSp;          /* Speed of maximum wind gust */ 
float   nMaxHailSize;               /* Hailstone diameter */ 
float   rsConvTemp;                /* Convection temperature */ 
} MALLTUAPROFILE, *PMALLTUAPROFILE;

```

3.4.4.3 Convective Conditions Structure

This structure holds additional information about observing station conditions.

```

typedef struct MALLTConvCond {
    char    szType;                  /* Type of station */ 
    int     nHeight;                 /* Station elevation */ 
    float   rsPressure;              /* Station pressure */ 
    float   rsTemperature;           /* Station temperature */ 
} MALLTCONVCOND, *PMALLTCONVCOND;

```

3.4.4.4 Evaporation Duct Height Structure

This structure stores the evaporation duct height and information used to calculate it.

```

typedef struct MALLTEvaporationHt {
    int     nEvapDuctHeight;          /* Evaporation duct height */ 
    float   rsWindSpeed;              /* Wind speed */ 
    float   rsSurfaceAirTemp;          /* Surface air temperature */ 
    float   rsSst;                   /* Sea surface temperature */ 
    short   sRelativeHumidity;        /* Relative humidity */ 
    float   rsDewPointTemp;            /* Dew point temperature */ 
} MALLTEVAPORATIONHT, *PMALLTEVAPORATIONHT;

```

3.4.4.5 Upper Air Profile Sounding Structure

This structure holds level-specific data for a single level of an upper air profile.

```
typedef struct MALLTUAProfileSounding {
    short sHeightType;          /* Type of level (SURFACE, STANDARD,      */
                               /* TROPOPAUSE, MAX WIND, SIGNIFICANT TEMP, */
                               /* SIGNIFICANT WIND, MISSING                */
    int   nHeight;              /* Height of current level                  */
    int   nMUnits;              /* Modified refractivity units             */
    short sRLType;              /* Refractive layer type code (1-5)        */
                               /* 1 = subrefractive, 2 = normal,           */
                               /* 3 = superrefractive, 4 = trapping,       */
                               /* 5 = undefined                          */
    float rsVerticalWindShear;  /* Vertical wind shear                     */
    int   nPotentialTemp;       /* Potential temperature                   */
    short sRichsNumber;         /* Richardson Number                      */
    short sTurbulenceProb;     /* Turbulence probability                 */
    char  szTurbulenceIntsty;  /* Turbulence intensity                  */
} MALLTUAPROFILESOUNDING, *PMALLTUAPROFILESOUNDING;
```

3.4.5 Structures Used for Rocketsonde Observations

3.4.5.1 Rocketsonde Report Structure

This structure holds data from a rocketsonde report.

```
typedef struct MALLTRocketReport{
    int   nNumSoundings;        /* Number of levels reported            */
    PMALLTROCKETSONDING pRS;    /* Pointer to level data structures */
} MALLTROCKETSONDEREPORT, *PMALLTROCKETSONDEREPORT;
```

3.4.5.2 Rocketsonde Sounding Structure

This structure holds data for a single level of a rocketsonde sounding.

```
typedef struct MALLTRocketSounding {
    int   nHeight;              /* Height of level (dependent on sHeightType) */
    int   nWindDirection;       /* Wind direction at level                  */
    float rsTemperature;        /* Temperature at level                   */
    short sHeightType;          /* Indicates units of rsHeight. 0 indicates */
                               /* meters, 1 hectopascals, 2 geopotential */
                               /* meters                                         */
    short sHeightUnit;          /* Unit identifier for height               */
    float rsWindSpeed;          /* Wind speed at level                    */
    float rsPressure;           /* Pressure at level                     */
    float rsDensity;            /* Air density at level                  */
} MALLTROCKETSONDING, *PMALLTROCKETSONDING;
```

3.4.6 Aerodrome Report Structures

3.4.6.1 METAR/SPECI Report Structure

This structure contains all data from a METAR or SPECI report.

```
typedef struct MALLTMETARReport {
    MALLTAERODROMELOUDS    clouds[3];           /* Pointer to cloud description */
    /* structures */                                */
    MALLTRUNWAYCONDITIONS  rw[10];              /* Pointer to runway conditions */
    /* */                                           */
    MALLTAERODROMEQP       qp[3];               /* Pointer to qualifier- */
    /* phenomena structures */                      */
    /* */                                           */
    int                    nNumRunways,          /* Number of runways reported */
    nNumClouds,             /* Number of cloud structures */
    nNumQPs;                /* Number of qualifier- */
    /* phenomenon structures */                     */
    /* */                                           */
    MALLTMETARFIELDS       mf;                  /* Data fields from report */
} MALLTMETARREPORT, *PMALLTMETARREPORT;
```

3.4.6.2 METAR/SPECI Fields Structure

This structure contains observed data from a METAR or SPECI report.

```
typedef struct MALLTMETARFields {
    char    szRemarks[32];           /* Remarks */
    /* */                           */
    int     nVisibility;            /* Visibility */
    /* */                           */
    int     nWindGusts;             /* Wind gust speed */
    /* */                           */
    char    szWindUnit[4];          /* Units in which wind speed is reported */
    /* */                           */
    char    szReportType[5];        /* Report type (METAR or SPECI) */
    /* */                           */
    float   rsWindSpeed;           /* Sustained wind speed */
    /* */                           */
    float   rsAirTemperature;      /* Air temperature */
    /* */                           */
    float   rsPressure;            /* Atmospheric pressure */
    /* */                           */
    float   rsDewPointTemp;         /* Dew point temperature */
    /* */                           */
    float   rsAltimeterSetting;    /* Altimeter setting */
    /* */                           */
    short   sWindDirection;         /* Direction from which wind is blowing */
    /* (degrees clockwise from true north) */
    /* */                           */
    short   sHorizontalViz;         /* Horizontal visibility */
    /* */                           */
    short   sMaxHorizontalViz;      /* Maximum horizontal visibility */
    /* */                           */
    short   sCCWindDirVar;          /* Direction of variable winds farthest */
    /* counter-clockwise from sWindDirection */
    /* */                           */
    short   sCWWindDirVar;          /* Direction of variable winds farthest */
    /* clockwise from sWindDirection */
    /* */                           */
    char    szVizDir[2];            /* Direction of sHorizontalViz (N,S,E,W) */
    /* */                           */
    char    szMaxVizDir[2];          /* Direction of sMaxHorizontalViz */
    /* */                           */
    char    szStationType[2];        /* Station type (manual or automated) */
    /* */                           */
    char    szAltimeterUnit[2];      /* Indicator for altimeter setting units */
    /* */                           */
    char    szWindVariability[2];    /* Wind variability indicator */
    /* */                           */
} MALLTMETARFIELDS, *PMALLTMETARFIELDS;
```

3.4.6.3 Aerodrome Clouds Structure

This structure describes the cloud conditions for a METAR/SPECI or TAF report.

```
typedef struct MALLTAAFCLOUDS {
    short sCloudHeight;            /* Altitude of base of cloud layer */
    /* */                           */
    char  szCloudAmount[4];          /* Cloud amount (FEW, SCT, BKN, OVC) */
    /* */                           */
    char  szType[4];                /* Type of clouds */
    /* */                           */
} MALLTAAFCLOUDS, *PMALLTAAFCLOUDS;
```

```
} MALLTAERODROMEClouds, *PMALLTAERODROMEClouds;
```

3.4.6.4 Aerodrome Qualifier-Phenomena Structure

This structure holds information about phenomena occurring at or near the observing station.

```
typedef struct MALLTQP { /* Structure for Qualifier-Phenomena(QP) */
    char szIntensity[4]; /* Intensity modifier (- for light, + for heavy, */
    /* VC for phenomenon in vicinity of station) */
    char szDescriptor[4]; /* Descriptor for phenomenon (WMO-306 Code Table */
    /* 4678) */
    char szPhenomena[4]; /* Weather phenomenon (WMO-306 Code Table 4678) */
} MALLTAERODROMEQP, *PMALLTAERODROMEQP;
```

3.4.6.5 Runway Conditions Structure

This structure holds information concerning conditions on a single runway.

```
typedef struct MALLTRunwayConditions {
    float rsMinVisualRange; /* Minimum visual range on runway */
    float rsMaxVisualRange; /* Maximum visual range on runway */
    char szRunwayDesig[2]; /* Runway designator (e.g. R, L, C) */
    short sRunwayNumber; /* Runway number */
    short sWindShear; /* Flag for presence of wind shear on runway */
    char szUnit; /* Unit of runway visual range */
    char szTendency[8]; /* Tendency of runway visual range values */
    /* U=increasing, D=decreasing, N=no change */
    char szVizQualifier[4]; /* Visibility qualifier */
} MALLTRUNWAYCONDITIONS, *PMALLTRUNWAYCONDITIONS;
```

3.4.6.6 TAF Report Structure

This structure holds data from a TAF report. TAF reports are unique in that they contain forecast as well as observed information.

```
typedef struct MALLTTAFAReport {
    PMALLAEROFORECAST pForecasts; /* Pointer to forecast structures */
    int nNumForecasts; /* Number of forecast structures */
    float rsWSWindSpeed; /* Wind Shear Speed */
    float rsWSLevel; /* Level at which wind shear occurs */
    short sWSDirection; /* Direction from station of wind */
    /* shear */
    char szWindUnit[4]; /* Units of wind shear speed */
    MALLTTAFFIELDS generalForecast; /* Current conditions */
} MALLTTAFAREPORT, *PMALLTTAFAREPORT;
```

3.4.6.7 TAF Fields Structure

This structure contains observed or forecast data from a TAF report.

```
typedef struct MALLTTAFFields {
    int nClouds, /* Number of clouds structures */
    int nQPs; /* Number of qualifier-phenomenon */
    /* structures */
    MALLTTAFCONDS conditions; /* Wind and visibility conditions */
    MALLTMETOCTIME beginningTime; /* Beginning time of forecast */
} MALLTTAFFIELDS;
```

```

MALLTMETOCTIME      endTime;          /* End time of forecast           */
MALLTAERODROMEQP    qp[3];           /* Qualifier-phenomenon structures */
MALLTAERODROMELOUDS clouds[3];       /* Cloud data structures          */
} MALLTAFIELDS, *PMALLTAFIELDS;

```

3.4.6.8 TAF Conditions Structure

This structure contains current or forecast wind and visibility information from a TAF.

```

typedef struct MALLTAFConds {
    float rsWindSpeed;          /* Wind speed                      */
    float rsVisibility;         /* Visibility                       */
    char szWindUnits[4];        /* Wind speed units indicator     */
    short sWindDirection;       /* Wind direction                  */
    char szVizUnits[2];         /* Visibility units indicator      */
} MALLTAFCONDS, *PMALLTAFCONDS;

```

3.4.6.9 Aerodrome Forecast Structure

This structure contains information related to a forecast (qualifiers for time of forecast validity, forecast trend, and probability of an occurrence, and forecast conditions).

```

typedef struct MALLTAeroForecast
{
    char      szFcstIndicator[4]; /* Time qualifier FM (from), TO (to), */
                           /* TL (until)                         */
    char      szTrend[6];        /* Trend qualifier PROB (probable,   */
                           /* TEMPO (temporarily),             */
                           /* BECMG (becoming)                */
    short     sProbability;     /* Probability (only set if       */
                           /* szTrend = PROB)                 */
    MALLTAFIELDS forecast,     /* Forecast conditions              */
    trend;                 /* Forecast trend                   */
} MALLTAEROFORECAST, *PMALLAEROFORECAST;

```

3.4.7 Surface Synoptic Observation Structures

3.4.7.1 Synoptic Observation Report Structure

This structure holds data from a surface synoptic observation.

```

typedef struct MALLTSynopticReport{
    int      nHasOcean;        /* Flag for ocean data in report */
    MALLTSYNOPTICSUMMARY synSum; /* Synoptic summary structure    */
    MALLTCLOUDDATA clouds;    /* Cloud data structure          */
    MALLTOCEANDATA ocean;     /* Ocean data structure          */
} MALLTSYNOPTICREPORT, *PMALLTSYNOPTICREPORT;

```

3.4.7.2 Synoptic Summary Structure

This structure holds summary information from a synoptic report.

```

typedef struct MALLTSynopticSummary {
    float rsWetBulbTemp;       /* Wet bulb temperature           */
    float rsPrsTendencyAmt;   /* Amount of pressure change over the
                               /* last 3 hours                  */
    float rsMaxTemp;          /* Maximum temperature during period */
} MALLTSYNOPTICSUMMARY, *PMALLTSYNOPTICSUMMARY;

```

```

float  rsMinTemp;           /* Minimum temperature during period      */
float  rsPrecipAmount;      /* Precipitation amount during period    */
float  rsPrecip24Hr;        /* Precipitation amount during last 24 hrs */
float  rsMoisture;         /* Dew point or relative humidity       */
                           /* (dependent on szMoistureType)        */
float  rsHeight;           /* Station height (GPH or height above MSL) */
float  rsPressure;          /* Atmospheric pressure at station       */
int    nHorzVisibility;     /* Horizontal visibility                 */
int    nSunshineDuration;   /* Duration of sunshine over past hour   */
int    nStationType;        /* Station type (manual or automated)   */
char   szSupplementary[80]; /* Pointer to supplementary information  */
                           /* about phenomena occurring at time of */
                           /* observation (WMO-306 Code Table 3778) */

short  sPastWxPrim;         /* Primary past weather indicator        */
short  sPastWxSecond;       /* Secondary past weather indicator       */
short  sPresentWx;          /* Present weather indicator             */
short  sIceAccrSource;      /* Source of ice accretion (WMO-306      */
                           /* Code Table 1751)                      */
short  sIceAccrRate;        /* Rate of ice accretion (WMO-306        */
                           /* Code Table 3551)                      */
short  sIceAccrThickness;   /* Thickness of ice accretion in cm       */
short  sPrecipPeriod;       /* Period over which precipitation occurred*/
                           /* isobaric level                       */
short  sMoistureType;       /* 0 if rsMoisture represents relative   */
                           /* humidity, 1 if dew point temperature  */
short  sWindSpeedSource;    /* Indicator of source of wind speed     */
                           /* measurement                          */
short  sPrsTendencyChar;   /* Characteristic of pressure change over */
                           /* the last 3 hours                     */

} MALLTSYNOPTICSUMMARY, *PMALLTSYNOPTICSUMMARY;

```

3.4.7.3 Cloud Data Structure

The CLOUDDATA structure holds information concerning the cloud heights, cloud types, and amount of cloud coverage observed.

```

typedef struct MALLTCLOUDDATA {
    float rsCloudHeight;      /* Height of cloud above mean sea level */
    float rsHeightAboveSfc;   /* Height of cloud above surface        */
    char  szCloudAmount[2];   /* Cloud amount from WMO-306 Table 2700 */
    char  szLowCloud[2];     /* Code figure for low cloud type from */
                           /* WMO-306 Table 0513                  */
    char  szMidCloud[2];     /* Code figure for middle cloud type from */
                           /* WMO-306 Table 0513                  */
    char  szHighCloud[2];    /* Code figure for high cloud type from */
                           /* WMO-306 Table 0509                  */
} MALLTCLOUDDATA, *PMALLTCLOUDDATA;

```

3.4.7.4 Ocean Data Structure

This structure holds ocean surface data from a ship synoptic report.

```

typedef struct MALLTOceanData {
    float rsSst;              /* Sea surface temperature               */
    float rsWaveHeight;        /* Significant wave height              */
    float rsWaveHeightInst;   /* Wave height measured by instruments */
    float rsWindWaveHeight;   /* Wind wave height                   */
} MALLTOceanData, *PMALLTOceanData;

```

```

float  rsPrimarySWHeight;      /* Primary swell wave height          */
float  rsSecondarySWHeight;    /* Secondary swell wave height        */
short   sPrimarySWDir;        /* Primary swell wave direction      */
short   sSecondarySWDir;      /* Secondary swell wave direction    */
short   sWavePeriod;          /* Significant wave period           */
short   sWavePeriodInst;      /* Wave period measured by instruments */
short   sPrimarySWPeriod;     /* Primary swell wave period         */
short   sSecondarySWPeriod;   /* Secondary swell wave period       */
char    szBearingEdge;        /* Bearing of principle ice edge     */
char    szConcentration;      /* Concentration or arrangement of sea
                                /* ice (WMO-306 Code Table 0639)    */
char    szDevStage;           /* Stage of development of sea ice   */
                                /* (WMO-306 Code Table 3739)        */
char    szLandOrigin;          /* Ice of land origin (Code Table 0439) */
char    szSituationTrend;     /* Present ice situation and trend   */
                                /* (WMO-306 Code Table 5239)        */
}
} MALLTOCEANDATA, *PMALLTOCEANDATA;

```

3.4.8 Ocean Observation Structures

3.4.8.1 Buoy Report Structure

This structure holds data from a buoy report.

```

typedef struct MALLTBuoyReport {
    MALLTBUOY             br;      /* Buoy surface data                  */
    PMALLTBUOYSOUNDING   pBUS;    /* Pointer to buoy sounding level structures */
    int                   nSoundings; /* Number of sounding levels reported */
} MALLTBUEYREPORT, *PMALLTBUEYREPORT;

```

3.4.8.2 Buoy Surface Data Structure

This structure contains surface weather data from a buoy report.

```

typedef struct MALLTbuoy{
    float   rsPressureTendAmt;    /* Amount of pressure change over last 3
                                    /* hours                               */
    float   rsMSLPressure;        /* Mean sea level pressure            */
    short   sWavePeriodByInst;    /* Wave period measured by instrument */
    float   rsMoisture;          /* Moisture value at level (dependent on
                                    /* szMoistureType)                   */
    float   rsWaterDepth;         /* Maximum water depth               */
    short   sDriftSpeed;          /* Drift speed of buoy               */
    short   sMoistureType;        /* Indicates how moisture is reported. 0
                                    /* indicates that rsMoisture reports
                                    /* relative humidity, 1 indicates dew
                                    /* point                             */
    char    szPressureTendChar[2]; /* Characteristic of pressure tendency */
} MALLTBUOY, *PMALLTBUOY;

```

3.4.8.3 Buoy Sounding Data Structure

This structure holds data from a single level of a sounding reported by a buoy.

```

typedef struct MALLTbuoySounding{
    float   rsDepth;             /* Depth of level                   */
    float   rsSalinity;           /* Salinity at level                */
} MALLTBUOYSOUNDING, *PMALLTBUOYSOUNDING;

```

```
    float rsWaterTemp;          /* Water temperature at level */  
} MALLTBUOYSOUNDING, *PMALLTBUOYSOUNDING;
```

3.4.8.4 Bathythermograph Report Data Structure

This structure holds data from a bathythermograph, Depth-Temperature-Salinity (DTS), sound velocimeter, or similar oceanographic profile.

```
typedef struct MALLTBathyReport {  
    MALLTBATHY           bt;  
    PMALLTBATHYSOUNDING pBAS;      /* Pointer to sounding level */  
                                    /* structures */  
    int                 nSoundings, /* Number of level structures */  
} MALLTBATHYREPORT, *PMALLTBATHYREPORT;
```

3.4.8.5 Bathythermograph Sounding Level Structure

This structure holds data from a single level of a bathythermograph, DTS, sound velocimeter, or similar oceanographic profile.

```
typedef struct MALLTbathySounding {  
    float rsDepth;            /* Depth */  
    float rsWaterTemp;        /* Water temperature */  
    float rsSalinity;         /* Salinity (parts per thousand) */  
    float rsCurrentSpeed;    /* Current speed */  
    float rsSoundSpeed;       /* Sound speed */  
    short sCurrentDir;        /* Current direction */  
} MALLTBATHYSOUNDING, *PMALLTBATHYSOUNDING;
```

3.4.8.6 Ocean Profile Summary Data Structure

This structure holds summary data for an ocean profile. An ocean profile may contain sounding data plus data calculated from the sounding.

```
*****  
*      Ocean Profile STRUCTURE  
*      This structure holds an OCEAN PROFILE Object  
*****  
typedef struct MALLTOceanProfile  
{  
    int             nObjectID;      /* Unique object identifier */  
    float           rsWindSpeed;    /* Wind Speed (m/s) */  
    long            lBottomDepth;   /* Bottom depth in meters */  
    long            lObservationTime; /* Epoch time of observation */  
    long            lPSource;       /* Ob source (BT, DSV, etc) */  
    long            lPState;        /* Indicator for state of */  
                                /* profile (MrgBT, MrgXSV, */  
                                /* BT, XSV, Forecast, Hist) */  
    long            lBottomFlag;    /* Flag indicating whether */  
                                /* profile extends to bottom */  
    long            lError;         /* Flag for error in data */  
    int             nGeoFlag;       /* Land-sea flag */  
    MALLTBOTLOSSDATA botLossData; /* Bottom loss data */  
    MALLTICECAPDATA iceCapData;  /* ICECAP data */  
    MALLTSSOTHERINFO ssOtherInfo; /* Other sound speed-related */
```

```

        /* data */  

long          lNumLevels;      /* Number of levels in */  

/* in sounding */  

PMALLTDTSSSPROFILE pLevels;  /* Pointer to sounding level */  

/* structures */  

float         rsRange;        /* Range to phenomenon */  

float         rsBearing;      /* Bearing to phenomenon */  

float         rsMldDay;       /* Mixed layer depth (day) */  

float         rsMldNight;     /* Mixed layer depth (night) */  

float         rsTopLayerDay;  /* Top of scattering layer */  

/* (day) */  

float         rsBotLayerDay;  /* Bottom of scattering */  

/* layer (day) */  

float         rsBotLayerNight; /* Bottom of scattering */  

/* layer (night) */  

int          nVssDets;       /* Number of volume scat- */  

/* tering detail sets */  

PMALLTVSSDET pVssDet;       /* Pointer to volume scat- */  

/* tering detail structures */  

} MALLTOCEANPROFILE, *PMALLTOCEANPROFILE;

```

3.4.8.7 Bottom Loss Data Structure

This structure holds data needed to calculate bottom loss for sound propagation calculations.

```

*****
*      BOTLOSS
*****
typedef struct MALLTBotLossData
{
    float   rsRange;        /* Range to bottom loss area */  

    float   rsBearing;      /* Bearing to bottom loss area */  

    long    lProv;          /* Bottom loss province */  

    float   rsRatio;        /* Ratio of sound speed in sediment to that in */  

/* water at the interface */  

    float   rsLThk;         /* Thin layer thickness in meters */  

    float   rsLDens;        /* Thin layer density in g per cubic cm */  

    float   rsSDens;        /* Sediment density in g per cubic cm */  

    float   rsSsGrad;       /* Initial sound speed gradient in dB/m */  

    float   rsSsCurv;       /* Sound speed profile curvature */  

    float   rsAttn;         /* Initial attenuation in sediment, dB/m/kHz */  

    float   rsAttnGr;       /* Attenuation gradient, dB/m/kHz/m */  

    float   rsReflec;       /* Basement reflection coefficient */  

    float   rsAttExp;       /* Frequency exponent for BLUG parameters */  

    float   rsSThick;       /* Sediment thickness in meters */  

    long    lMgs;           /* MGS region number */  

} MALLTBOTLOSSDATA, *PMALLTBOTLOSSDATA;

```

3.4.8.8 ICECAP Data Structure

This structure contains data used in the ICECAP ice prediction program.

```

*****
*      ICECAP
*****

```

```
typedef struct MALLTIcecapData
{
    float    rsIceLoss;    /* Transmission loss in ice, dB/m */
    float    rsKeelSp;     /* Keel spacing in meters */
    float    rsRange;      /* Range to ice edge */
    float    rsBearing;    /* Bearing to ice edge */
    float    rsUir;        /* Under ice roughness */
    float    rsRf;         /* Ridge frequency */
} MALLTICECAPDATA, *PMALLTICECAPDATA;
```

3.4.8.9 Volume Scattering Detail Structure

This structure contains volume scattering parameters for a single level.

```
*****
*      VSSDET
*****
typedef struct MALLTVssDet
{
    float    rsVssFreqs;   /* Volume scattering frequencies */
    float    rsVssDay;     /* Volume scattering strength (day) */
    float    rsVssNight;   /* Volume scattering strength (night) */
} MALLTVSSDET, *PMALLTVSSDET;
```

3.4.8.10 Ocean Profile Sounding Structure

This structure holds sounding data from a single level of an ocean profile.

```
*****
*      OCEAN PROFILE
*****
typedef struct MALLTDtsssProfile
{
    float    rsDepth;       /* Depth of level in meters */
    float    rsTemp;        /* Water temperature in degrees C */
    float    rsSalin;       /* Salinity in parts per thousand */
    float    rsSoundSpeed;  /* Sound speed in meters per second */
} MALLTDTSSSPROFILE, *PMALLTDTSSSPROFILE;
```

3.4.8.11 Sound Speed-Related Data Structure

This structure holds sound speed-related data from an ocean profile.

```
*****
*      SSOTHERINFO
*****
typedef struct MALLTSSOtherInfo
{
    float    rsSspm;        /* Mean sound speed in profile, m/s */
    float    rsDsca;        /* Deep sound channel axis depth, meters */
    float    rsDscb;        /* Depth at sound channel bottom, meters */
} MALLTSSOTHERINFO, *PMALLTSSOTHERINFO;
```

```

float rsSscb; /* Sound speed at channel bottom, m/s */ 
float rsSmin; /* Minimum sound speed in profile, m/s */ 
float rsIwarn; /* Warning flag for message */ 
float rsSld; /* Sonic layer depth, meters */ 
float rsTsld; /* Temperature at sonic layer depth, degrees C */ 
float rsSsld; /* Sound speed at sonic layer depth, m/s */ 
float rsSmax; /* Maximum sound speed in profile, m/s */ 
float rsCd; /* Critical depth, meters */ 
float rsTop; /* Depth of top of channel, meters */ 
float rsDe; /* Depth excess, meters */ 
float rsVe; /* Velocity excess, m/s */ 
float rsChnmag; /* Sound channel magnitude */ 
float rsTgin; /* In-layer temperature gradient, deg C/100 m */ 
float rsTgbl; /* Below-layer temperature gradient, deg C/100 m */ 
float rsSgin; /* In-layer sound speed gradient, m/s/100 m */ 
float rsSbgl; /* Below-layer sound speed gradient, m/s/100 m */ 
float rsCoFreq; /* Cut-off frequency, kHz */ 
} MALLTSSOTHERINFO, *PMALLTSSOTHERINFO;

```

3.4.9 Query Structures

3.4.9.1 LLT Observation Catalog Query Structure

This structure is used to provide input criteria for a catalog search of LLT observations in the database.

```

typedef struct MALLTDSDirQuery {
    int      nSiteID,          /* Numeric identifier for observing site */ 
            nDataCategory, /* Code designating state of data relative to */ 
                           /* its original form: 0 = base, 1 = edited, */ 
                           /* 2 = derived */ 
            nQualityIndicator, /* Is the QC flag set? */ 
            nMinTime,         /* Earliest observation time desired (epoch time) */ 
            nMaxTime;         /* Latest observation time desired (epoch time) */ 
            nMinIngestTime /* Begin ingest time of interest */ 
            nMaxIngestTime /* Ending ingest time of interest */ 
    short    sObType,           /* Observation type */ 
            sObSubtype;        /* Observation subtype */ 
    char     szSiteName[16]; /* String identifier of observing station */ 
    MALLTGEOUNDS geoArea; /* Geographic area */ 
} MALLTOBCATALOGQUERY, *PMALLTOBCATALOGQUERY;

```

3.4.9.2 Geographic Boundaries Structure

This structure contains the geographic boundaries of a query area.

```

typedef struct MALLTGeoBounds {
    float   rsNorthLat; /* Northernmost latitude of area */ 
    float   rsSouthLat; /* Southernmost latitude of area */ 
    float   rsEastLon;  /* Easternmost longitude of area */ 
    float   rsWestLon;  /* Westernmost longitude of area */ 
} MALLTGEOUNDS, *PMALLTGEOUNDS;

```

3.4.10 Catalog Structures

3.4.10.1 LLT Observation Catalog Structure

This structure holds catalog information about a single observation, returned from a catalog query.

```
typedef struct MALLTObCatalog {
    MALLTOBREFERENCE obRef;           /* Observation reference info */
    char szObTypeName[16];           /* Observation type */
    char szObSubTypeName[32];         /* Observation subtype */
    MALLTOBTYPES obType;             /* Observation type and subtype */
    char szReceiptMethod[32];         /* Circuit over which data were */
                                     /* received */
    char szSiteName[16];              /* Observing station ID */
    int mSiteID,                      /* Observing station numeric ID */
        nDataCategory,                /* Code designating state of */
                                     /* data relative to its original */
                                     /* form; 0 = base, 1 = edited, */
                                     /* 3 = derived */
        nQualityIndicator,            /* QC flag information */
        nDatasetID;                  /* Dataset identifier */
    MALLTGEOPPOINT geoLoc;           /* Geographic location of ob */
    MALLTMETOCTIME baseTime;          /* Observation time */
    ingestTime;                      /* Time at which ob was ingested */
} MALLTOBCATALOG, *PMALLTOBCATALOG;
```

3.4.10.2 LLT Forecast Catalog Structure

This structure contains catalog information about a single LLT Forecast (TAF), returned from a catalog query.

```
typedef struct MALLTFCstObCatalog {
    MALLTOBREFERENCE obRef;           /* Observation reference info */
    char szObTypeName[16];           /* Observation type name */
    char szObSubTypeName[32];         /* Observation subtype name */
    MALLTOBTYPES obType;             /* Type and subtype of ob */
    char szReceiptMethod[32];         /* Circuit over which data were */
                                     /* received */
    char szSiteName[16];              /* Reporting station ID */
    int nDataCategory,                /* Code designating state of */
                                     /* data relative to its original */
                                     /* form; 0 = base, 1 = edited, */
                                     /* 2 = derived */
        nQualityIndicator;            /* QC flags */
        nDatasetID;                  /* Dataset identifier */
    MALLTGEOPPOINT geoLoc;           /* Geographic location */
    MALLTMETOCTIME baseTime;          /* Issue time of forecast */
    MALLTMETOCTIME beginvalidtime;    /* Beginning valid time */
    MALLTMETOCTIME endvalidtime;      /* Ending valid time */
    MALLTMETOCTIME ingestTime;        /* Time at which ob was ingested */
} MALLTFCSTOBBCATALOG, *PMALLTFCSTOBBCATALOG;
```

3.4.11 Linked List Structure

This structure contains a linked list of catalog data or observation data structures. The type of structure pointed to by the data member is determined by the type of data being returned in the linked list.

```
typedef struct tagMALLTLinkedList {
    char *next;          /* Pointer to next element */
    char *prev;          /* Pointer to previous element */
    int nEntryCount;    /* Number of entries in the list */
    short sIsDataField; /* Data field flag */
} MALLTLINKEDLIST, *PMALLTLINKEDLIST;
```

3.4.12 LLT Observation Return Structure

Each of the LLT observation APIs returns this structure containing status data. The *nStatus* field will contain a zero upon successful completion of the API call. If the field is set to 1, there is an SQL error, and the *szSQLState* field must be examined. Any value of *nStatus* greater than 1.

```
typedef struct tagMALLTRet
{
    int nStatus;           /* nStatus = 0 ==> Success */
                           /* Non-zero nStatus indicates an error */
    char szSQLState[6];   /* First 2 characters represent the class, last 3 */
                           /* characters the subclass where an SQL error */
                           /* occurred */
    char szErrorMessage[290]; /* Textual description of error */
} MALLTRET, *PMALLTRET;
```

(This page intentionally left blank.)

4 CONNECT APIs

The connect APIs are used to establish or disestablish a connection to the database server on which the MDLLT database resides.

Information about each API is presented in manual page format as follows:

NAME

Function Name – Provides a brief description of the function.

SYNOPSIS

Presents the calling syntax for the routine, including the declarations of the arguments and the return type. Also lists the necessary include files for each routine.

INPUT PARAMETERS

Describes each of the input parameters used by the function.

OUTPUT PARAMETERS

Describes each of the output parameters used by the function.

DESCRIPTION

Describes what the function does and what events or side effects it causes.

RETURN VALUES

Describes what the function returns.

NOTE

1. Provides any applicable notes about the function.

SEE ALSO

Provides a reference to related functions.

Examples showing the proper usage of the APIs are presented in the *LLT Observation API Programming Manual*, referenced in Section 2. All API calling sequences are provided in the “C” programming language.

4.1 MALLTConnect

NAME

MALLTConnect – Connects the calling application to the default LLT observation database.

SYNOPSIS

```
#include "MALLTAPI.h"  
MALLTRET MALLTConnect( void );
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTConnect function initializes internal data structures and establishes a connection to the default observation database environment.

RETURN VALUES

MALLTConnect returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTE

1. A database connection must be established using MALLTConnect or MALLTRemoteConnect before other MALLT API functions (or SQL) that act on the database can succeed. If the application connects to multiple databases during execution, MALLTRemoteConnect should be used instead of MALLTConnect.

SEE ALSO

MALLTDisconnect, MALLTRemoteConnect, MALLTSetConnection.

4.2 MALLTDisconnect

NAME

MALLTDisconnect – Disconnects the calling application from the currently active LLT observation database.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTDisconnect ( void );
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTDisconnect function disconnects the calling application from the currently active database session.

RETURN VALUES

MALLTDisconnect returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTE

1. This function should be called for orderly termination of the active database session. For applications requiring disconnection from a dormant database session, a connection name must be specified using the MALLTRemoteDisconnect function.

SEE ALSO

MALLTConnect, MALLTRemoteDisconnect, MALLTSetConnection.

4.3 MALLTRemoteConnect

NAME

MALLTRemoteConnect – Connects the application to the specified LLT database environment.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTRemoteConnect (    char *szDatabase,
                                char *szConnectionName );
```

INPUT PARAMETERS

szDatabase – Name of the database environment. Valid forms of name are:

- “dbname”
- “@dbservername”
- “dbname@dbservername”

If NULL is passed in, the application will connect to the default database.

szConnectionName – Name of the connection to the database. This is a unique name associated with a given connection to the database. Any alphanumeric string up to 32 characters is valid. If NULL is passed in, the default connection name is used. **Most users will set this variable to NULL.** It is provided to users who may need to manage connections to multiple databases within an application.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTRemoteConnect function initializes internal data structures and establishes a connection to the specified observation database environment, beginning the database session. The connection becomes the current connection for the calling application.

RETURN VALUES

MALLTRemoteConnect returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTE

1. When remote or multiple databases are being used, a call to MALLTRemoteConnect must succeed, and the connection be active, before subsequent MALLT API functions (or SQL) can succeed. Applications connecting to multiple databases should call MALLTSetConnection to set a database environment from dormant to active.

SEE ALSO

MALLTRemoteDisconnect, MALLTSetConnection

4.4 MALLTRemoteDisconnect

NAME

MALLTRemoteDisconnect – Disconnects the application from the specified LLT observation database environment.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTRemoteDisconnect (char *szConnection );
```

INPUT PARAMETERS

szConnection – Name of connection given when the MALLTRemoteConnect function was called. If NULL was passed in via MALLTRemoteConnect function, NULL must also be passed into the MALLTRemoteDisconnect function.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTRemoteDisconnect function disconnects the calling application from the specified database environment, ending the database session.

RETURN VALUES

MALLTRemoteDisconnect returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTE

1. This function should be called for orderly termination of each specified database session started with MALLTRemoteConnect.

SEE ALSO

MALLTRemoteConnect, MALLTSetConnection.

4.5 MALLTSetConnection

NAME

MALLTSetConnection – Sets the active connection to the specified LLT observation database.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTSetConnection( char *szConnection );
```

INPUT PARAMETERS

szConnection – Name of a desired connection previously specified in a call to the **MALLTRemoteConnect** function.

OUTPUT PARAMETERS

None.

DESCRIPTION

The **MALLTSetConnection** function is used to switch between named connections to the LLT observation database. It deactivates the current connection and activates the connection named in **szConnection**.

RETURN VALUES

MALLTSetConnection returns the status of the call in the **MALLTRET** structure. The **nStatus** field of **MALLTRET** will equal 0 if the call succeeded – any other value indicates an error, which will be described in the **szErrorMessage** field. See Section 3.4.12 for information about the **MALLTRET** structure.

NOTE

1. This function must be called to make a dormant database connection active. Calls to other **MALLT** API functions (or SQL) act only on the active database.

SEE ALSO

MALLTRemoteConnect, **MALLTRemoteDisconnect**.

(This page intentionally left blank.)

5 RETRIEVAL APIs

This section describes the APIs that are used to retrieve data from the MDLLT database.

5.1 MALLTCatalog

NAME

MALLTCatalog – Queries the active database for a catalog of LLT observations based on selection criteria.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTCatalog (      MALLTOBCATALOGQUERY  dsCatCriteria,
                           int                  *pnNumItems,
                           PMALLTLINKEDLIST    pCatList );
```

INPUT PARAMETERS

dsCatCriteria – Pointer to a query structure containing selection criteria. A query structure is provided for each observation type and subtype.

OUTPUT PARAMETERS

pnNumItems – Pointer to the number of records found matching the selection criteria.
pCatList – Pointer to a linked list of catalog records summarizing observations that met the given selection criteria.

DESCRIPTION

The MALLTCatalog function is a generic routine to retrieve a catalog of observations from the active LLT observation database. The pCatList linked list points to a set of PMALLTOBCATALOG or PMALLTFCSTOBBCATALOG structures. The number of observations matching the criteria is returned via the pnNumItems pointer. The obType field within each structure can be used to determine which structure it is.

RETURN VALUES

MALLTCatalog returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method replaces the MALLTCatalogObs method from the developer release. Applications calling MALLTCatalogObs should be modified to call MALLTCatalog instead, as MALLTCatalogObs will be phased out in the next release.
2. The desired database environment must be active before this function can succeed. Also see sections on structures for a fuller description of the input, output, and return parameters.
3. The linked list returned by MALLTCatalog should be freed using the MALLTFreeLL method when no longer required.

SEE ALSO

MALLTConnect, MALLTRemoteConnect, MALLTSetConnection, MALLTFreeLL.

5.2 MALLTCatalogObs

NAME

MALLTCatalogObs – Queries the active database for a catalog of LLT observations based on selection criteria.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTCatalogObs ( MALLTCATALOGQUERY    dsCatCriteria,
                           int                  *pnNumItems,
                           PMALLTLINKEDLIST   pCatList );
```

INPUT PARAMETERS

dsCatCriteria – Pointer to a query structure containing selection criteria. A query structure is provided for each observation type and subtype.

OUTPUT PARAMETERS

pnNumItems – Pointer to the number of records found matching the selection criteria.

pCatList – Pointer to a linked list of catalog records summarizing observations that met the given selection criteria.

DESCRIPTION

The MALLTCatalogObs function is a generic routine to retrieve a catalog of observations from the active LLT observation database. The number of items matching the criteria is returned via the pnNumItems pointer. The pCatList linked list points to a set of PMALLTOBCATALOG or PMALLTFCSTOBBCATALOG structures. The obType field within each structure can be used to determine which structure it is.

RETURN VALUES

MALLTCatalogObs returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method has been superseded by MALLTCatalog. Applications calling MALLTCatalogOb should be modified to call MALLTCatalog instead, as MALLTCatalogObs will be phased out in the next release.
2. The desired database environment must be active before this function can succeed. Also see sections on structures for a fuller description of the input, output, and return parameters.

3. The linked list returned by MALLTCatalogObs should be freed using the MALLTFreeLL method when no longer required.

SEE ALSO

MALLTConnect, MALLTRemoteConnect, MALLTSetConnection.

5.3 MALLTGetByID

NAME

MALLTGetByID – Retrieves data for a specified LLT observation.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTGetByID ( MALLTOREFERENCE obRef,
                         PMALLTObservation pAnOb ) ;
```

INPUT PARAMETERS

obRef – Specifies the object ID and table name of the desired observation.

OUTPUT PARAMETERS

pAnOb – Pointer to a MALLTObservation structure containing the LLT observation record.

DESCRIPTION

The MALLTGetByID function retrieves an observation record based on the observation reference criteria.

RETURN VALUES

MALLTGetByID returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method supersedes the MALLTGetOb method from the developer release. Applications calling MALLTGetOb should be modified to call MALLTGetByID instead, since MALLTGetOb will be phased out in the next release.
2. The desired database environment must be active before this function can succeed. A catalog list of existing observation records can be obtained by calling MALLTCatalog.

SEE ALSO

MALLTCatalog, MALLTConnect, MALLTRemoteConnect, MALLTSetConnection

5.4 MALLTGetOb

NAME

MALLTGetOb – Retrieves data for a specified LLT observation.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTGetOb ( MALLTOREFERENCE obRef,
                        PMALLTObservation pAnOb ) ;
```

INPUT PARAMETERS

obRef – Specifies the object ID and table name of the desired observation.

OUTPUT PARAMETERS

pAnOb – Pointer to a MALLTObservation structure containing the LLT observation record.

DESCRIPTION

The MALLTGetOb function retrieves an observation record based on the observation reference criteria.

RETURN VALUES

MALLTGetOb returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method has been superseded by the MALLTGetByID method. Applications calling MALLTGetOb should be modified to call MALLTGetByID instead, since MALLTGetOb will be phased out in the next release.
2. The desired database environment must be active before this function can succeed. A catalog list of existing observation records can be obtained by calling MALLTCatalog.

SEE ALSO

MALLTCatalog, MALLTConnect, MALLTRemoteConnect, MALLTSetConnection

5.5 MALLTGetByQuery

NAME

MALLTGetByQuery – Retrieves a set of LLT observations that match specified query criteria.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTGetByQuery ( MALLTOBCATALOGQUERY dsCatCriteria,
                           int *pnNumRecs
                           PMALLTLINKEDLIST pObs );
```

INPUT PARAMETERS

dsCatCriteria – Pointer to a query structure containing selection criteria. A query structure is provided for each observation type and subtype.

OUTPUT PARAMETERS

pnNumRecs – Pointer to the number of records returned.
pDetList – Pointer to linked list of observations that met the given selection criteria.

DESCRIPTION

The MALLTGetByQuery function is a generic routine to query the LLT observation database and return observations meeting the query criteria. The data portion of the linked list will contain a pointer to a PMALLTOBSERVATION structure. The integer point pnNumRecs reports the number of observations that were returned.

RETURN VALUES

MALLTGetByQuery returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method supersedes the MALLTGetObs method from the developer release. Applications calling MALLTGetObs should be modified to call MALLTGetByQuery instead, since MALLTGetObs will be phased out in the next release.
2. The desired database environment must be active before this function can succeed.
3. The linked list created should be freed using the MALLTFreeLL method when no longer required.

SEE ALSO

MALLTConnect, MALLTCatalog, MALLTGetByID, MALLTSetConnection, MALLTFreeLL

5.6 MALLTGetObs

NAME

MALLTGetObs – Retrieves a set of LLT observations that match specified query criteria.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTGetObs ( MALLTOBCATALOGQUERY dsCatCriteria,
                        int *pnNumRecs
                        PMALLTLINKEDLIST pDetList );
```

INPUT PARAMETERS

dsCatCriteria – Pointer to a query structure containing selection criteria. A query structure is provided for each observation type and subtype.

OUTPUT PARAMETERS

pnNumRecs – Pointer to the number of records returned.
pDetList – Pointer to linked list of observations that met the given selection criteria.

DESCRIPTION

The MALLTGetObs function is a generic routine to query the LLT observation database and return observations meeting the query criteria. The data portion of the linked list will contain a pointer to a PMALLTOBSERVATION structure.

RETURN VALUES

MALLTGetObs returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method has been superseded by MALLTGetByQuery. Applications calling MALLTGetObs should be modified to call MALLTGetByQuery instead, since MALLTGetObs will be phased out in the next release.
2. The desired database environment must be active before this function can succeed.
3. The linked list created should be freed using the MALLTFreeLL method.

SEE ALSO

MALLTConnect, MALLTCatalog, MALLTGetByID, MALLTSetConnection, MALLTFreeLL

5.7 MALLTGetStationByArea

NAME

MALLTGetStationByArea – This function receives a geographic area, queries the database for stations residing there, then returns a list of the stations found.

SYNOPSIS

```
#include "MALLTAPI.h"

MALLTRET MALLTGetStationByArea ( MALLTGEOUNDS theArea, MALLTLINKEDLIST
                                  *pStations, int *pnNumberStations, int
                                  nQueryFor );
```

INPUT PARAMETERS

- | | |
|----------------------|--|
| MALLTGEOUNDS theArea | – The latitude/longitude-bounded search area. |
| int nQueryFor | – The type of station sought in the query. Possible values include:
1 = WMO stations
2 = ICAO stations
3 = WMO stations + ICAO stations
4 = Buoy stations
5 = WMO stations + Buoy stations
6 = ICAO stations + Buoy stations
7 = All stations |

OUTPUT PARAMETERS

- | | |
|----------------------------|--|
| MALLTLINKEDLIST *pStations | – A pointer to the list of MALLTSTATIONINFO structures detailing the stations found. |
| int *pnNumberStations | – A pointer to the number of stations retrieved. |

DESCRIPTION

A query of the database is made using the area input in `theArea` structure and the station type specified in the `nQueryFor` input. If corresponding stations were found, a pointer to a linked list of station information structures is returned using `pStations`, and a pointer to the number of stations retrieved is returned in the `pnNumberStations` argument. If no stations were found, the value pointed to by `pnNumberStations` will be zero.

RETURN VALUES

MALLTGetStationByArea returns the status of the call in the MALLTRET structure. The `nStatus` field of MALLTRET will equal 0 if the call succeeded. A non-zero value indicates that

an error occurred. The error is further described in the szErrorMessage field of the structure. See Section 3.4.12 for more MALLTRET structure information.

NOTE

1. The values for WMO, ICAO, and Buoy station types are defined in malltmacros.h.. The station information structure, MALLTSTATIONINFO, is declared in malltcommon.h. These files are automatically included through the *MALLTAPI.h* header file.

SEE ALSO

MALLTGetStationByID.

5.8 MALLTGetStationByID

NAME

MALLTGetStationByID – Receives a station ID and returns detailed station information.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTGetStationByID ( MALLTGENERICID stationID,
                                PMALLTSTATIONINFO pStation );
```

INPUT PARAMETERS

MALLTGENERICID stationID – The ID of the station of interest.

5.8.1 OUTPUT PARAMETERS

PMALLTSTATIONINFO pStation – A pointer to the MALLTSTATIONINFO structure of station information.

DESCRIPTION

This function receives the stationID for a WMO, ICAO, or Buoy, then queries the database for a match. If a match was found, the function returns a pointer pStation to a station information structure (containing station name, location, elevation, type, country of residence, etc.). If the specified station was not found, the function return status will not be 0.

RETURN VALUES

MALLTGetStationByID returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded. A non-zero value indicates that an error occurred. The error is further described in the szErrorMessage field of the structure. See Section 3.4.12 for more MALLTRET structure information.

NOTE

1. The station type values are defined in the malltmacros.h file. These files are automatically included through the MALLTAPI.h header file.

SEE ALSO

MALLTGetStationByArea.

5.9 Observation Types

In each of the APIs described above, the user must input an observation type as an input-calling parameter. There are seven basic types of observations:

1. Upper Air Wind observations – winds observed in the atmosphere at levels above the surface, typically by radiosondes.
2. Upper Air Temperature observations – temperatures (air temperature and dew point temperature) observed in the atmosphere at levels above the surface, typically by radiosondes.
3. Rocketsonde observations – similar to upper air temperature reports, but made by a rocket-borne sensor.
4. Synoptic observations – Observations taken at specific times each day at sites all over the world to create a “snapshot” of global weather conditions.
5. Ocean observations – Observations of temperature and sometimes other parameters (sound speed, salinity, current) versus depth in the ocean.
6. Aircraft observations – Observations of atmospheric conditions collected by aircraft. These are generally “spot” observations, collected at a specific latitude, longitude, altitude, and time, of temperatures, winds, and other conditions such as airframe icing.
7. Aerodrome observations and forecasts – Observations/forecasts produced by airports for civil and military aviation use. These differ from the other observation types in that they usually include forecasts of future conditions as well as observations of current conditions.

Seven “C” Language macros have been defined in the **malltypes.h** include file, one for each observation type above as follows:

```
MALLT_UPPERAIR_WIND  
MALLT_UPPERAIR_TEMP  
MALLT_ROCKET_OB  
MALLT_SYNOPTIC_OB  
MALLT_OCEAN_OB  
MALLT_AIRCRAFT_OB  
MALLT_AERODROME_OB
```

5.10 Observation Subtype

Further granularity for identifying observation data is provided through a subtype. Subtypes, as the name suggests, are associated with an observation data type. “C” Language macros have been defined in the **malltypes.h** include file for observation data types and subtypes. Some subtypes are common to all data types. Some subtypes are unique to specific data types. The following is a list of observation subtypes:

- Ocean observation subtypes:

MALLT_BATHY_REPORT – Report of bathythermograph observation

MALLT_BUOY_REPORT – Report from an oceanographic buoy

MALLT_OCEAN_PROFILE – Ocean profile (may contain calculated data)

- Aerodrome observation subtypes:

MALLT_METAR_REPORT – Normal or hourly report in METAR format

MALLT_SPECI_REPORT – Special report in METAR format

MALLT_TAF_REPORT – TAF

- Subtypes common to upper air (TEMP and WIND), Rocketsonde, and Synoptic Observation Data:

MALLT_SHIP – Report from a ship

MALLT_FIXED_LAND – Report from a fixed land platform

MALLT_MOBILE_LAND – Report from a mobile land platform

- Upper Air TEMP subtypes:

MALLT_UAT_DROP – Report collected by dropsonde

MALLT_UAT_PROFILE – Upper air temperature profile (may contain calculated data)

- Aircraft observation subtypes:

MALLT_AIREP – Report in aircraft report (AIREP) format

MALLT_PIREP – Pilot report (PIREP)

(This page intentionally left blank.)

6 DATA MANAGEMENT APIs

6.1 MALLTDeleteByID

NAME

MALLTDeleteByID – Deletes the LLT observation that matches the specified observation reference.

SYNOPSIS

```
#include "MALLTAPI.h"  
MALLTRET MALLTDeleteByID ( MALLTOBREFERENCE obRef );
```

INPUT PARAMETERS

ObRef – Specifies the object ID and table name of the observation to delete.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTDeleteByID function deletes the LLT observation specified in the input observation reference structure.

RETURN VALUES

MALLTDeleteByID returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method supersedes the MALLTDeleteOb method from the developer release. Applications calling MALLTDeleteOb should be modified to call MALLTDeleteByID instead, since MALLTDeleteOb will be phased out in the next release.
2. The desired database environment must be active before this function can succeed.

SEE ALSO

MALLTConnect, MALLTCatalog, MALLTRemoteConnect, MALLTSetConnection

6.2 MALLTDeleteOb

NAME

MALLTDeleteOb – Deletes the LLT observation that matches the specified observation reference.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTDeleteOb ( MALLTOREFERENCE obRef );
```

INPUT PARAMETERS

ObRef – Specifies the object ID and table name of the observation to delete.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTDeleteOb function deletes the LLT observation specified in the input observation reference structure.

RETURN VALUES

MALLTDeleteOb returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method has been superseded by MALLTDeleteByID. Applications calling MALLTDeleteOb should be modified to call MALLTDeleteByID instead, since MALLTDeleteOb will be phased out in the next release.
2. The desired database environment must be active before this function can succeed.

SEE ALSO

MALLTConnect, MALLTCatalog, MALLTRemoteConnect, MALLTSetConnection

6.3 MALLTIgest

NAME

MALLTIgest – Adds the specified LLT observation data to the database.

SYNOPSIS

```
#include "MALLTAPI.h"  
MALLTRET MALLTIgest ( PMALLTOBSERVATION pAnOb );
```

INPUT PARAMETERS

pAnOb – Pointer to the referenced observation data structure containing the new data.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTIgest function inserts the input LLT observation data into the database.

RETURN VALUES

MALLTIgest returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method supersedes the MALLTIgestOb method from the developer release. Applications calling MALLTIgestOb should be modified to call MALLTIgest instead, since MALLTIgestOb will be phased out in the next release.
2. The desired database environment must be active before this function can succeed.
3. Tables are created dynamically. The names of tables are created by concatenating a predefined prefix with the current coordinated universal (Z) time. Child tables will follow the same convention but will also have a 1- to 4-letter suffix. The table on the next page maps the datatype/subtype to the table prefix.

obtypename	obs subtypename	obtypeid	obs subtype	tableprefix
AERODROME	METAR	7	3	SA
AERODROME	SPECI	7	4	SP
AERODROME	TAF	7	5	FT
AIRCRAFT	PIREP	6	13	PR
AIRCRAFT	AIREP	6	12	AR
OCEAN_OB	Buoy	5	2	ZZ
OCEAN_OB	Bathy	5	1	JJ
OCEAN_OB	Ocean Profile	5	14	OP
ROCKET	Ship	3	6	SS
ROCKET	Fixed Land	3	7	RR
SYNOPTIC	Mobile Land	4	8	OO
SYNOPTIC	Fixed Land	4	7	AA
SYNOPTIC	Ship	4	6	BB
UPPERAIR TEMP	Profile	2	10	UP
UPPERAIR TEMP	Drop	2	9	XX
UPPERAIR TEMP	Mobile Land	2	8	II
UPPERAIR TEMP	Fixed Land	2	7	TT
UPPERAIR TEMP	Ship	2	6	UU
UPPERAIR WIND	Mobile Land	1	8	EE
UPPERAIR WIND	Fixed Land	1	7	PP
UPPERAIR WIND	Ship	1	6	QQ

SEE ALSO

[MALLTConnect](#), [MALLTRemoteConnect](#), [MALLTSetConnection](#)

6.4 MALLTIgestOb

NAME

MALLTIgestOb – Adds the specified LLT observation data to the database.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTIgestOb ( PMALLTOBSERVATION pAnOb );
```

INPUT PARAMETERS

pAnOb – Pointer to the referenced observation data structure containing the new data.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTIgestOb function inserts the input LLT observation data into the database.

RETURN VALUES

MALLTIgestOb returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method has been superseded by MALLTIgest. Applications calling MALLTIgestOb should be modified to call MALLTIgest instead, since MALLTIgestOb will be phased out in the next release.
2. The desired database environment must be active before this function can succeed.
3. Tables are created dynamically. The names of tables are created by concatenating a predefined prefix with the current coordinated universal (Z) time. Child tables will follow the same convention but will also have a 1- to 4-letter suffix. The table that maps the datatype/subtype to the table prefix can be found in Section 6.3.

SEE ALSO

MALLTConnect, MALLTRemoteConnect, MALLTSetConnection

6.5 MALLTUpdateByID

NAME

MALLTUpdateByID – Updates an existing LLT observation record using the data specified.

synopsis

```
#include "MALLTAPI.h"
MALLTRET MALLTUpdateByID ( MALLTOREFERENCE obRef,
                            PMALLTOBSERVATION pAnOb ) ;
```

INPUT PARAMETERS

obRef – Specifies the object ID and table name of the observation to update.

pAnOb – Pointer to the observation data structure containing the update data.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTUpdateByID function takes as inputs a MALLTOREFERENCE structure identifying the table and record to be updated, and a PMALLTOBSERVATION structure containing the new observation data. The record identified is updated with the new data supplied.

RETURN VALUES

MALLTUpdateByID returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method supersedes the MALLTUpdateOb method from the developer release. Applications calling MALLTUpdateOb should be modified to call MALLTUpdateByID instead, since MALLTUpdateOb will be phased out in the next release.
2. The desired database environment must be active before this function can succeed. The function will fail if the observation specified does not exist.

SEE ALSO

MALLTConnect, MALLTRemoteConnect, MALLTSetConnection.

6.6 MALLTUpdateOb

NAME

MALLTUpdateOb – Updates an existing LLT observation record using the data specified.

SYNOPSIS

```
#include "MALLTAPI.h"
MALLTRET MALLTUpdateOb ( MALLTOREFERENCE obRef,
                           PMALLTOBSERVATION pAnOb );
```

INPUT PARAMETERS

obRef – Specifies the object ID and table name of the observation to update.

pAnOb – Pointer to the observation data structure containing the update data.

OUTPUT PARAMETERS

None.

DESCRIPTION

The MALLTUpdateOb function takes as inputs a MALLTOREFERENCE structure identifying the table and record to be updated, and a PMALLTOBSERVATION structure containing the new observation data. The record identified is updated with the new data supplied.

RETURN VALUES

MALLTUpdateOb returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTES

1. This method has been superseded by MALLTUpdateByID. Applications calling MALLTUpdateOb should be modified to call MALLTUpdateByID instead, since MALLTUpdateOb will be phased out in the next release.
2. The desired database environment must be active before this function can succeed. The function will fail if the observation specified does not exist.

SEE ALSO

MALLTConnect, MALLTRemoteConnect, MALLTSetConnection.

6.7 MALLTDeleteByQuery

NAME

MALLTDeleteByQuery – Purges all records from the database meeting given criteria.

SYNOPSIS

```
#include "MALLTAPI.h"

MALLTRET MALLTDeleteByQuery ( MALLTOBCATALOGQUERY dsCatCriteria )
```

INPUT PARAMETERS

dsCatCriteria – MALLTOBCATALOGQUERY structure containing purge criteria.

OUTPUT PARAMETERS

None.

DESCRIPTION

MALLTDeleteByQuery purges from the database all records that meet the criteria contained in the input MALLTOBQUERY structure.

RETURNS

MALLTDeleteByQuery returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTE

None.

SEE ALSO

None.

7 MALLT UTILITY METHODS AND FUNCTIONS

7.1 MALLTFreeLL

NAME

MALLTFreeLL – Frees an LLT observation linked list.

SYNOPSIS

```
#include "MALLTAPI.h"  
MALLTRET MALLTFreeLL ( MALLTLINKEDLIST *pLL );
```

INPUT PARAMETERS

pLL – Pointer to the linked list to be freed.

OUTPUT PARAMETERS

None.

DESCRIPTION

MALLTFreeLL frees a linked list that has been returned by a call to a query operation. Methods that return a linked list include MALLTCatalog, MALLTCatalogObs, MALLTGetByQuery, and MALLTGetObs.

RETURNS

MALLTFreeLL returns the status of the call in the MALLTRET structure. The nStatus field of MALLTRET will equal 0 if the call succeeded – any other value indicates an error, which will be described in the szErrorMessage field. See Section 3.4.12 for information about the MALLTRET structure.

NOTE

None.

SEE ALSO

MALLTCatalog, MALLTCatalogObs, MALLTGetByQuery, MALLTGetObs

7.2 Null Functions

MALLT provides functions to set various data types to NULL, and to evaluate whether entries for various data types are NULL. The Set To Null functions should be used to indicate that a field within an observation was not reported, or when performing a retrieve, to indicate fields that should be excluded from a query. These functions are listed below.

7.2.1 Set To NULL Functions

Each of these functions returns a NULL value for the given datatype.

```
double MALLTsetDouble2NULL (void)  
float MALLTsetFloat2Null (void)  
short MALLTsetShort2Null (void)  
int MALLTsetInt2Null (void)  
long MALLTsetLong2Null (void)
```

Usage example:

Performing a catalog query for which you are not interested in the WMO ID of the reporting site:

```
MALLTOBCATALOGQUERY MOCQ;  
MOCQ.nSiteID = MALLTsetInt2Null();
```

The above sets the nSiteID member to NULL.

To indicate string values as NULL, set the first character [0] of the string to zero.

7.2.2 Functions to Evaluate Whether a Value is NULL

Each of these functions returns 1 (true) if the input parameter is NULL, and 0 (false) if it is not NULL.

```
int MALLTisDoubleNull ( double rd )  
int MALLTisFloatNull ( float rs )  
int MALLTisShortNull ( short s )  
int MALLTisIntNull ( int n )  
int MALLTisLongNull ( long l )
```

8 NOTES

8.1 Glossary of Acronyms

AESS	Allied Environmental Support System
AIREP	Aircraft Report
API	Application Program Interface
APIRM	API Reference Manual
COE	Common Operating Environment
DII	Defense Information Infrastructure
DTS	Depth-Temperature-Salinity
GCCS	Global Command and Control System
IC4ISR	Integrated Command, Control, Communications, Computer, and Intelligence Surveillance Reconnaissance
IMOSS	Interim Mobil Oceanographic Support System
IP	Installation Procedures
JMCIS	Joint Maritime Command Information System
JMS	Joint METOC Segment
LLT	Latitude-Longitude-Time
MALLT	LLT Observation API Segment
MDLLT	LLT Observation Database Segment
METOC	Meteorology and Oceanography

MIDDS	Meteorological Integrated Data Display System
NITES	Navy Integrated Tactical Environmental Subsystem
PIREP	Pilot Report
PM	Programming Manual
PS	Performance Specification
QP	Qualifier-Phenomena
RDBMS	Relational Database Management System
SQL	Structured Query Language
SWEAT	Severe Weather Threat
TAF	Terminal Aerodrome Forecast
TEDS	Tactical Environmental Data System
TESS(NC)	Tactical Environmental Support System Next Century